KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT INGENIEURSWETENSCHAPPEN
DEPARTEMENT COMPUTERWETENSCHAPPEN
AFDELING INFORMATICA
Celestijnenlaan 200 A — B-3001 Leuven

ALGEBRAIC AND LOGICAL STUDY OF CONSTRUCTIVE PROCESSES IN KNOWLEDGE REPRESENTATION

Jury :
Prof. Dr. ir. H. Van Brussel, voorzitter
Prof. Dr. M. Denecker, promotor
Prof. Dr. D. De Schreye, promotor
Prof. Dr. ir. M. Bruynooghe
Prof. Dr. J. Denef
Prof. Dr. M. Gelfond (Texas Tech University, VS)
Prof. Dr. T. Eiter (Technische Universität Wien, Oostenrijk)

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de ingenieurswetenschappen

door

Joost VENNEKENS

U.D.C. 681.3*I2

Mei 2007

D/2007/7515/49
ISBN 978-90-5682-815-8

Algebraic and Logical Study of Constructive Processes in Knowledge Representation

Joost Vennekens

Abstract

Constructive processes play an important role in knowledge representation. Indeed, there are many formal languages whose semantics can be characterized using fixpoint criteria, that simulate, for instance, human thought processes or mathematical construction principles. Such processes can be studied in an abstract, algebraic way. This allows common properties of such languages to be examined in general, without committing to any particular syntax or semantics. In a first part of this thesis, we examine two topics in this way: first, we look at modularity of theories and, second, we consider certain transformations that extend the vocabulary of a theory to simplify some of its formulas. In both cases, we find that single algebraic theorem about constructive processes suffices to derive (partial) generalizations of a number of different existing results for logic programs, autoepistemic logic, and default logic.

In a second part of the thesis we examine the link between constructive processes and the concept of causality. We observe that causality has an inherent dynamic aspect, i.e., that, in essence, causal information concerns the evolution of a domain over time. Motivated by this observation, we construct a new representation language for causal knowledge, whose semantics is defined explicitly in terms of constructive processes. This is done in a probabilistic context, where the basic steps that make up the process are allowed to have non-deterministic effects. We then show that a theory in this language defines a unique probability distribution over the possible outcomes of such a process. This result offers an appealing explanation for the usefulness of causal information and links our explicitly dynamic approach to more static causal probabilistic modelling languages, such as Bayesian networks. We also show that this language, which we have constructed to be a natural formalization of a certain kind of causal statements, is closely related to logic programming. This result demonstrates that, under an appropriate formal semantics, a rule of a normal, a disjunctive or a certain kind of probabilistic logic program can be interpreted as a description of a causal event.

Acknowledgments

Writing a PhD is an educational activity. It has taught me a lot, not least of all about my own shortcomings. I have learned, for instance, that my writing has an unfortunate tendency towards the unreadably dense, while my thoughts are often rushed and imprecise, if not plainly false (I should never again be allowed within 20 feet of the adverb “clearly”). If, despite these faults, I have managed to produce a Ph.D. that is of any interest, this is largely due to the guidance I have received along the way. I am especially grateful to my promoter Marc Denecker for his untiring persistence on clarity and precision, without which this text would have undoubtedly been a lot less read-worthy. Marc was always great at noticing flaws (in my writing, arguments, or general character) long before anyone else, and also at continuing to remind me of them long after everybody else had stopped caring. Moreover, many of the ideas inside these covers have originated from him, or came up during one of the many interesting coffee breaks we have shared together.

While Marc has had without a doubt the greatest influence on my work, I have been fortunate enough to also benefit from the guidance of two other senior members of the DTAI group. First, there was my promoter Danny De Schreye, who, together with Sofie Verbaeten, oversaw my first steps into academia and originally introduced me to the topic of probabilistic logic programming. Second, I also thank Maurice Bruynooghe for being a very helpful coauthor, as well as a source of wisdom on a variety of topics, ranging from the scientific over the financial to the practical.

I would also like to thank the other members of my jury, Michael Gelfond, Thomas Eiter, Jan Denef, and the chairman Herman Van Brussel, for taking time out of their busy schedule to read this work and attend my defense. I am particularly grateful to Jan for agreeing to be part of my guidance committee at such a late stage in the Ph.D. and would also like to thank Bart De Decker for having been his predecessor during the first four years. Jan’s presence on my jury is especially fitting, since it were his two courses on mathematical logic, more than any other, that have motivated me through my years of study. I also owe Michael a special debt of gratitude for having me in his research group at Texas Tech—and his home—for a very pleasant two months. Even though we did not get around to any formal collaboration, I learned a lot from being there. Also, quite a bit of the research in Chapter 4 was actually motivated by a couple of off-hand remarks made during the very first kr-lab meeting I attended there.

Another flaw that Marc has on occasion spotted in me (he is really quite proficient) is that of, as he phrased it, “intellectual laziness”. Honestly, he was just being kind, since the qualification “intellectual” could easily be left out altogether. If, despite my

slothful tendencies, I have nevertheless managed to be in the office during somewhat respectable working hours, this was probably in large part due to my desire to miss neither the regular ten o'clock coffee break, nor the one at four. For creating such a pleasant working environment, I am especially grateful to my office mates Stephen Bond, the much acclaimed Author of Interactive Fiction, and ir. Maarten Mariën, whom I gladly forgive for being better than I am at all things we ever did together; to Álvaro Cortés-Calabuig, whom I was also very glad to have as a conference-going buddy over the years; and to Anneleen Van Assche, who was already my office mate long before Maarten and Stephen made it fashionable.

During the past few years, I have been fortunate enough to have had a number of friends who were also in the process of obtaining their Ph.D., which means I always had someone on hand with whom I could share the frustrations of academic life. In particular, I would like to thank Pieter Bellens, Pieter Vermeulen, Wouter Castryck, and Bert Wouters for their pleasant and uplifting company. Thanks for keeping me rooted in reality, on the other hand, go to Kris Van Rompaye and Jeroen Deceuninck—their stories about real life have always been a great inspiration for trying my best to avoid it. Finally, I would also like to thank Pauwel De Dijn, Stefaan Neerinck, and Stijn Adriaens for doing their part in keeping the country supplied of consumer goods at, respectively, cheap, reasonable, and reasonably cheap prices.

To anyone who knows them, it will go without saying that, throughout these four years and a half, my parents, Mieke Vermeulen and Eddy Vennekens, were always there to help me in whatever way they could. Since even things that go without saying should, I believe, on occasion be said, I would like to take this opportunity to say that I am very grateful for their support. Dank jullie wel!

Finally, I would also like to thank Marijke Verbruggen: for having already been my girl for many years, for soon also becoming my wife, and for—

well, she knows what for.

Joost Vennekens
Leuven, May 2007

I am not here in the happy position of a mineralogist who shows his audience a rock-crystal: I cannot put a thought in the hands of my readers with the request that they should examine it from all sides. Something in itself not perceptible by sense, the thought is presented to the reader—and I must be content with that—wrapped up in a perceptible linguistic form. The pictorial aspect of language presents difficulties. The sensible always breaks in and makes expressions pictorial and so improper. So one fights against language, and I am compelled to occupy myself with language although it is not my proper concern here.

– Gottlob Frege, *Der Gedanke*

Contents

List of Figures

List of Symbols

$(\Delta_i)_{i\in I}$	A partition of rule set Δ, page 42
$(\Sigma_i)_{i\in I}$	Splitting for a propositional logic program, page 34
$(At_i)_{i\in I}$	A partition of set of domain atoms, page 42
$[\sigma]_D$	Equivalence class of C-selection w.r.t. $D \subseteq C$, page 135
$\mathcal{A}t^F_{\mathbf{P}}$	Set of domain atoms using pre-interpretation F and predicates $\mathbf{P}$, page 12
$\mathcal{B}^c$	The square of $\mathcal{W}^c_\Sigma$, page 46
$\mathcal{B}_\Sigma$	Square of $\mathcal{W}_\Sigma$, page 17
CP_B	CP-logic translation of Bayesian network B, page 112
$\tilde{\mathcal{B}}_{\Sigma'}$	Square of $\tilde{\mathcal{W}}_{\Sigma'}$, page 44
$\tilde{\mathcal{B}}^c_{\Sigma'}$	Square of $\tilde{\mathcal{W}}^c_{\Sigma'}$, page 46
$\tilde{\mathcal{B}}^{\downarrow c}_{\Sigma'}, \tilde{\mathcal{B}}^{\uparrow c}_{\Sigma'}$	The square of, resp., $\tilde{\mathcal{W}}^{\downarrow c}_{\Sigma'}$ and $\tilde{\mathcal{W}}^{\uparrow c}_{\Sigma'}$, page 77
$\mathcal{C}_A$	Partial stable operator of approximation A, page 9
$\mathcal{E}$	Mapping of nodes of a C-process to events in C, page 95
$\mathcal{I}$	Mapping of nodes of a Σ-process to interpretations, page 94
$\mathcal{P}$	Mapping of nodes of a Σ-process to probabilities, page 95
$\mathcal{D}_T$	Immediate consequence operator for autoepistemic logic, page 18
$\mathcal{D}^u_T$	Second element of the pair produced by $\mathcal{D}_T$, page 18
$\Delta[\varphi/P]$	The result of replacing φ by P in Δ, page 67
Δ	A rule set, page 10
δ	In the context of predicate introduction, the rule set defining the new predicate, page 60

$B_F(\varphi)$	Base for φ, page 38
$B^{\mathcal{O}}_{(O_1,O_2)}$	A $\mathcal{O}$-base for a formula, page 40
$\tilde{\mathcal{D}}_T$	Stratifiable version of $\mathcal{D}_T$, defined on $\tilde{\mathcal{B}}_{\Sigma'}$, page 47
$\tilde{\mathcal{D}}_{T'}$	The fixpoint extension corresponding to T', page 77
$\mathcal{H}_{I,(P,S)}(\varphi)$	Truth value of φ in autoepistemic logic, page 18
$\mathcal{I}_\Sigma$	Set of interpretations for propositional alphabet Σ (= 2^Σ), page 17
κ	In the context of CP-logic, a mapping from events to time points, page 102
κ	In the context of autoepistemic logic, mapping from $\tilde{\mathcal{W}}_{\Sigma'}$ to $\mathcal{W}_\Sigma$, page 44
$\overline{\kappa}$	Mapping from $\tilde{\mathcal{B}}_{\Sigma'}$ to $\mathcal{B}_\Sigma$, page 44
$\mathcal{L}^F_{\mathbf{P}}$	Set of all interpretations for predicates $\mathbf{P}$ that extend F, page 14
λ	A temporal assignment (or stratification) for a CP-theory, page 101
$\leq$	A partial order on a set S; also the point-wise extension of $\leq$ to S^2; also the product order of a product lattice, page 9
$\leq_k$	The knowledge order, page 12
$\leq_p$	The precision order, page 9
$\leq_t$	The truth order, page 12
$\lfloor\cdot\rfloor, \lceil\cdot\rceil \ldots$	Various projection functions, page 59
$\varphi\langle P, S\rangle$	Result of replacing modal literals by their truth value in (P, S), page 19
$\varphi\langle U, V\rangle_i$	Result of evaluating model literals in alphabet $\cup_{j\prec i}\Sigma_j$ in (U, V), page 50
$\mathcal{GL}_\Delta$	Gelfond-Lifschitz operator for rules set Δ(= $C^{\downarrow}_\Delta$), page 14
$\models_s$	A pair is a partial stable model of a rule set, page 15
$\models_w$	A pair is the well-founded model of a rule set, page 15
μ_C	The instance based semantics for an LPAD, page 117
$[T_i]$	Modally separated theory corresponding to T_i, page 52
ν	A four-valued or three-valued interpretation, page 12
ν_s	Three-valued interpretation constructed in node s of a C-process, page 104
Π	A propositional logic program, page 33
π_B	Semantics of a Bayesian network B, page 110

π_C	Probability distribution defined by CP-theory C, page 98
$\preceq$	Order on the index set of a product, page 23
ψ	In predicate introduction for autoepistemic logic, the least modal literal containing the formula $K\varphi$ that is to be replaced, page 76
$\mathcal{T}$	A probabilistic Σ-process, page 94
$\mathcal{T} \models_X C$	$\mathcal{T}$ is an execution model of C in X, page 95
$f\|_{A'}$	The restriction of $f : A \to B$ to $A' \subseteq A$, page 7
$\mathcal{R}$	Mapping of nodes of a C-process to the set of events that have not yet happened, page 131
$\rightsquigarrow$	A dependency relation, page 30
$\rightsquigarrow_\Delta^{(O_1,O_2)}$	Dependency relation for rule set Δ given pair of interpretations (O_1, O_2) for $Op(\Delta)$, page 40
$\rightsquigarrow_\Delta^F$	Dependency relation for rule set Δ given pre-interpretation F, page 39
$\rightsquigarrow_\Delta^{ob}$	Obvious dependency relation for rule set Δ, page 38
$\rightsquigarrow_\Pi$	Obvious dependency relation for Π, page 34
$\mathcal{S}_C$	The set of all selections for CP-theory C, page 116
Σ	A first-order alphabet, page 10
$\sigma, \rho, \ldots$	Selections for a CP-theory, page 116
$\Sigma^p, \Sigma^f, \Sigma^o$	Predicate, function and object symbols of alphabet Σ, page 10
Σ_B	Logical vocabulary for Bayesian network B, page 110
τ	Isomorphism between four-valued interpretation and pairs of interpretations, page 12
$\mathcal{T}_\Delta^{(O_1,O_2)}$	Four-valued immediate consequence operator for a rule set, page 14
$\mathcal{V}_3, \mathcal{V}_4$	Three- or four-valued truth values, page 12
$\mathcal{W}_\Sigma$	Set of possible worlds structures for propositional alphabet Σ (= $2^{\mathcal{I}_\Sigma}$), page 17
$\mathcal{W}_\Sigma^c$	Set of consistent possible world structures (all $Q \neq \{\}$), page 46
$\tilde{\mathcal{W}}_{\Sigma'}$	Product lattice $\otimes_{i\in I}\mathcal{W}_{\Sigma_i}$, page 44
$\tilde{\mathcal{W}}_{\Sigma'}^c$	Set of all $\tilde{Q}$, such that $\forall i\ Q(i) \neq \{\}$, page 46
$\tilde{\mathcal{W}}_{\Sigma'}^{\downarrow c}, \tilde{\mathcal{W}}_{\Sigma'}^{\uparrow c}$	The set of all $\binom{X}{U}$ for which, resp., $U \neq \{\}$ and $X \neq \{\}$, page 77

$At(\varphi)$	Atoms appearing in autoepistemic formula φ, page 16
$At_O(\varphi)$	Objective atoms of φ, page 16
$B^{(x\ y)}$	For operator B on $(L_1 \times L_2)^2$, the operator $\lambda u, v.\lfloor B(\begin{smallmatrix} x & y \\ u & v \end{smallmatrix})\rfloor$, page 60
$B_{(u\ v)}$	For operator B on $(L_1 \times L_2)^2$, the operator $\lambda x, y.\lceil B(\begin{smallmatrix} x & y \\ u & v \end{smallmatrix})\rceil$, page 60
$body_{At}(r)$	Ground atoms 'in' the body of a CP-event, page 93
$body^+_{At}(r)$	Ground atoms appearing positively 'in' the body of r, page 106
$body^-_{At}(r)$	Ground atoms appearing negatively 'in' the body of r, page 106
C^σ	An instance of an LPAD, page 116
$C^\uparrow_A$, $C^\downarrow_A$	Upper and lower stable operators of approximation A, page 9
C_v	Constant symbol for value v of a node in a Bayesian network, page 110
$Def(\Delta)$	Defined predicates in rule set Δ, page 10
$f \sqcup g$	The extension of $f : A \to B$ with $g : C \to D$, page 7
F	In predicate introduction for autoepistemic logic, the original formula, page 76
$F, G, \ldots$	Pre-interpretations for an alphabet, page 11
F_p	In predicate introduction for AEL, the formula that defines the new atom p, page 76
$head^*(r)$	The set of (h_i, α_i) in $head(r)$ extending with $(\emptyset, 1 - \sum \alpha_i)$, page 116
$head_{At}(r)$	Ground atoms in the head of a CP-event, page 93
I	A (well-founded) partially ordered set, used as index, page 23
$I, J, \ldots$	Interpretations for first-order alphabet, page 11
m	Konolige transformation from default to autoepistemic logic, page 20
$Mod(T)$	Classical models of propositional theory T, page 19
$O_1, O_2, \ldots$	Interpretations for open predicates, page 14
O^u_i	Component on level i given u of stratifiable O, page 25
O^x_J	Recombination of O, page 32
$Op(\Delta)$	Open predicates in rule set Δ, page 10
$P, S, \ldots$	Possible world structures, page 17
P_n	Predicate symbol for node n of a Bayesian network, page 110

T_Δ^O	Two-valued immediate consequence operator, page 14
$t_\lambda(r)$	The interval in which a CP-event r should happen, page 101
$U, V, \ldots$	Interpretations of $\bigcup_{j \prec i} \Sigma_j$, page 35
$u, v, \ldots$	Elements of $L\|_{\prec i}$ for product lattice L, page 24
$U_\Delta^{(O_1,O_2)}$	First element of the pair produced by $\mathcal{T}_\Delta^{(O_1,O_2)}$, page 14
V_s	Operator used to characterize ν_s for a node s in a C-process, page 132

Chapter 1

Introduction

The main focus of this text is on the role of constructive processes in knowledge representation. In the area of non-monotonic reasoning, we find many logics whose semantics can be defined in terms of fixpoint constructs that simulate, e.g., human thought processes, actual physical processes, or even mathematical principles such as that of definition by induction. Many salient properties of such processes can be elegantly described in the framework of *approximation theory* (Denecker, Marek, and Truszczyński 2000). This is an algebraic fixpoint theory, which has been shown to capture in a natural way certain families of semantics for knowledge representation languages such as logic programming, default logic, and autoepistemic logic. Approximation theory provides an appealing setting in which to study properties of these languages. First, the algebraic nature of the theory often allows general results to be proven in a clean, compact, and elegant manner, which exposes the essence of the property without getting bogged down in syntactical details. Second, by proving a single theorem in approximation theory, one immediately obtains results for a number of different logics and, typically, also for an entire family of semantics for these logics. As such, the tedious task of reproving essentially the same property in slightly different circumstances can be avoided. In a first part of this text, we study two important knowledge representation properties in approximation theory.

The first topic is that of *compositionality* or *modularity*, i.e., we address the question of how to split a theory in such a way that the set of its components is, in some sense, equivalent to the theory as a whole. This question is important for a variety of theoretical and practical purposes. For instance, modularity results can be used to speed up computational tasks, to reduce the problem of analyzing a theory to the easier problem of analyzing its parts, or to show that updates of an existing knowledge base do not affect certain already established properties. We study this topic at the abstract, algebraic level of lattice operators in approximation theory and examine conditions under which it is possible to split such an operator into a number of smaller operators, while preserving certain of its fixpoints. We apply this result to logic programming, default logic, and autoepistemic logic, showing that, in each of these cases, our results extend existing theorems.

The second topic is that of *predicate introduction*, i.e., the common transformation

of introducing a new symbol to abbreviate some complex formula. One might do this, for instance, to make a theory more readable and compact by singling out a subformula that appears in a number of different locations, or to transform it into some useful normal form. Once again, we study this topic at the level of approximation theory, by examining conditions under which it is possible to extend a lattice operator to a larger lattice, in such a way that certain of its fixpoints are preserved. We then show how this result applies to autoepistemic logic and to logic programming, where it significantly generalizes existing results, by also allowing the new symbols to be defined recursively.

In both of these cases, approximation theory will make it possible to define natural algebraic counterparts to the relevant logical concepts, which then allows concrete results for different languages and different semantics to be derived in an easy and uniform way. Recently, Truszczyński has performed a similar study of the concept of strong equivalence in the setting of approximation theory (Truszczyński 2006). We see here an emerging picture of approximation theory as a general, abstract framework for the study of knowledge representation properties of logics with a fixpoint semantics.

The second part of this text concerns the role of causality in probabilistic modelling. Here, too, constructive processes are important. Let us consider, for instance, the statement "smoking causes cancer." Intuitively, this means that the act of having a cigarette will initiate some sequence of events within the human body and that this process might eventually lead to the development of a cancer. One of our central observations is that whenever we make such a causal statement, we are, at least implicitly, doing so in the context of the dynamic evolution of a domain. This motivates us to view such statements as describing the causes and effects of an *event*, of something that *happens*. We also consider events that are non-deterministic, which allows us to represent knowledge such as: "Smoking *might* cause cancer and the probability of this is 0.2." We then develop the knowledge representation language of *CP-logic*, in which a domain can be modeled by a set of such statements.

Our analysis of the intuitive meaning of these statements leads in a natural way to a formal semantics that is based on probabilistic processes. One of our key technical results is that, even though a single theory may generate any one of a number of different possible probabilistic processes, all of these will induce precisely the same probability distribution over their final states. This is an interesting property, because, typically, we are not really interested in the actual details of the evolution of a domain, but simply care about the probability of this leading to a certain result. For instance, we might not be interested in if or when a person decides to smoke a cigarette, but only want to know what the probability is of this person getting cancer. Our result shows that causal information in the form of a set of CP-events suffices to know which possible outcomes will occur with which probability.

Our study of causality in this explicitly dynamic context offers additional insight into Pearl's seminal work on this topic (Pearl 2000). We identify the concept of a CP-event as a unit of causal information, that is more basic than the parents-child relation underlying causal Bayesian networks. In this way, we get a more flexible and fine-grained representation of causal events, which allows more straightforward, compact, and elaboration tolerant models of causal knowledge. Moreover, we are able to support and clarify Pearl's claims on the robustness of causal knowledge and its importance for achieving compact representations, by showing that it captures precisely those aspects

of the behaviour of a probabilistic process that are relevant for its final outcome, while allowing irrelevant details to be ignored.

We also compare these causal probabilistic processes to the kind of the processes that are used to define various fixpoint semantics for logic programs. To this end, we define a probabilistic extension of logic programs, called *logic programs with annotated disjunctions*, and prove that it is essentially equivalent to CP-logic. This result makes explicit the connection between causal reasoning and logic programming that has long been implicitly present in this field and shows that, under certain semantics, a rule of a normal or disjunctive logic program can be interpreted as a description of a causal event. Moreover, it also helps to clarify the position of our logic among related work on probabilistic logic programming and offers a causal interpretation for, among others, the independent choice logic (Poole 1997).

In summary, the main contributions of this text are as follows. First, we have shown that the methodology of studying constructive processes in the algebraic framework of approximation theory offers a viable way of deriving interesting properties of different logics in a general and uniform way. Second, we have also shown that constructive processes are important for a correct understanding of a certain kind of causal statements, because they offer a natural way of formalizing the dynamic aspect inherent to them.

1.1 Structure of the text

This text is structured as follows:

- Chapter 2 summarizes a number of concepts and results from work by Denecker, Truszczyńsky and Marek on approximation theory.
- Chapter 3 investigates the topic of modularity in approximation theory and applies the resulting algebraic theorem to logic programs, autoepistemic logic and default logic. Part of the work presented in this chapter was published in (Vennekens, Gilis, and Denecker 2006; Vennekens, Gilis, and Denecker 2004b; Vennekens, Gilis, and Denecker 2004a; Vennekens and Denecker 2005).
- Chapter 4 studies the topic of predicate introduction. Again, this is first done at the abstract level of approximation theory and these algebraic results are then applied to logic programming and autoepistemic logic. Part of the work presented in this chapter was published in (Wittocx, Vennekens, Mariën, Denecker, and Bruynooghe 2006; Vennekens, Mariën, Wittocx, and Denecker 2007a; Vennekens, Mariën, Wittocx, and Denecker 2007b).
- Chapter 5 starts the second part of this text, which studies the importance of constructive processes for modelling causality. Concretely, in this chapter, we develop the language of Causal Probabilistic logic (CP-logic). Part of this work was published in (Vennekens, Denecker, and Bruynooghe 2006). We also relate CP-logic to logic programming, by first defining a probabilistic extension of disjunctive logic programs and then proving that this is equivalent to CP-logic. Part of this work was published in (Vennekens, Verbaeten, and Bruynooghe 2004).

- Chapter 6 contains the proofs of the results stated in Chapter 5.
- Finally, Chapter 7 presents our conclusions.

Part I

Algebraic study of logics with fixpoint semantics

Chapter 2

The approximation theory framework

Approximation theory is an algebraic fixpoint theory for arbitrary (non-monotone) operators, that generalizes all main semantics of a number of non-monotonic logics. As such, it allows properties of these different semantics for all of these logics to be studied in a uniform way. In this chapter, we will first summarize the mathematics behind approximation theory and then show how logic programming, default logic and autoepistemic logic fit into this framework.

The material in this chapter is a summary of work by Denecker, Truszczyński and Marek (Denecker, Marek, and Truszczyński 2000; Denecker, Marek, and Truszczyński 2004; Denecker, Marek, and Truszczyński 2003).

2.1 Formal preliminaries

In this section, we introduce the basic concepts used in approximation theory. Section 2.1.1 summarizes a number of well known definitions and results from lattice theory, while Section 2.1.2 contains an overview of approximation theory itself.

2.1.1 Sets, functions, orders, lattices and operators

For a function $f : A \to B$ and a subset A' of A, we denote by $f|_{A'}$ the restriction of f to A', i.e., $f|_{A'} : A' \to B : a' \mapsto f(a')$. If f, g are functions $f : A \to B$, $g : C \to D$ and the domains A and C are disjoint, we denote by $f \sqcup g$ the function from $A \cup C$ to $B \cup D$, such that for all $a \in A$, $(f \sqcup g)(a) = f(a)$ and for all $c \in C$, $(f \sqcup g)(c) = g(c)$. We call such a $f \sqcup g$ an *extension* of f.

A binary relation θ on a set S is *well-founded* if it has no infinite descending chains, i.e., if there exist no infinite sequences $(x_i)_{i\in\mathbb{N}}$ such that, for all $i \in \mathbb{N}$, $x_i \theta x_{i-1}$ and $x_i \neq x_{i-1}$. A binary relation $\leq$ on a set S is a *partial order* if it is reflexive, transitive and anti-symmetric. In this case, the pair $\langle S, \leq \rangle$ is called a *poset*. For a subset R of S, an element $m \in R$ is *minimal* if there is no $m' \in R$, such that $m' \leq m$. It can

be shown that a poset is well-founded iff every non-empty subset of S has a minimal element. A partial order on S is *total* if every two elements $x, y \in S$ are comparable, i.e., $x \leq y$ or $x \geq y$. If $\leq$ is a total order, then it is well-founded iff each non-empty subset of S has a unique minimal element.

For each subset R of S, an element l of S, such that $l \leq r$ for all $r \in R$ is a *lower bound* of R. An element g in S such that g is a lower bound of R and for each other lower bound l of R, $l \leq g$, is called the *greatest lower bound*, denoted $glb(R)$, of R. Similarly, an element u such that for each $r \in R$, $u \geq r$ is an *upper bound* of R and if such an upper bound is less or equal to each other upper bound of R, it is the *least upper bound* $lub(R)$ of R.

A pair $\langle L, \leq \rangle$ is a *lattice* if $\leq$ is a partial order on the non-empty set L, such that every two elements x, y of L have a greatest lower bound $glb(\{x, y\})$ and a least upper bound $lub(\{x, y\})$. A lattice $\langle L, \leq \rangle$ is *complete* if each subset L' of L has a greatest lower bound $glb(L')$ and least upper bound $lub(L')$. A complete lattice has a minimal (or *bottom*) element $\bot$ and a maximal (or *top*) element $\top$. Often, we will not explicitly mention the partial order $\leq$ of a lattice $\langle L, \leq \rangle$ and simply speak of the lattice L.

An *operator* is a function $O : L \to L$ from a lattice L to itself. An element $x \in L$ is a *prefixpoint* of O if $x \geq O(x)$, a *fixpoint* if $x = O(x)$ and a *postfixpoint* if $x \leq O(x)$. Such an operator O is *monotone* if for each $x, y \in L$, such that $x \leq y$, $O(x) \leq O(y)$. If O is a monotone operator on a complete lattice, then for every postfixpoint y, there exists a least element in the set of all prefixpoints x of O for which $x \geq y$. This least prefixpoint greater than y of O is also the least fixpoint greater than y of O. Moreover, it can be constructed by successively applying O to y, i.e., as the least upper bound of the sequence $(O^n(y))_{n \in \mathbb{N}}$. In particular, because $\bot$ is a trivial postfixpoint, O has a least prefixpoint which is equal to its least fixpoint and which can be constructed by successive application of O to $\bot$.

2.1.2 Approximation theory

Approximation theory is a general fixpoint theory for arbitrary operators, which generalizes ideas found in, among others, (Baral and Subrahmanian 1991), (Ginsberg 1988) and (Fitting 1989). Our presentation of this theory is based on (Denecker, Marek, and Truszczyński 2000). However, we will introduce a slightly more general definition of approximation. For a comparison between approximation theory and related approaches, we refer to (Denecker, Marek, and Truszczyński 2000) and (Denecker, Marek, and Truszczyński 2003).

Let $\langle L, \leq \rangle$ be a lattice. An element (x, y) of the square L^2 of the domain of such a lattice, can be seen as denoting a (possibly empty) interval $[x, y] = \{z \in L \mid x \leq z \leq y\}$. By reflexivity and transitivity of $\leq$, the interval corresponding to a pair (x, y) is non-empty iff $x \leq y$. Such pairs are called *consistent*. By anti-symmetry of $\leq$, an interval of the form $[x, x]$ contains precisely one element, namely x itself. Elements (x, x) of L^2 are called *exact*. The set of all exact elements of L^2 forms a natural embedding of L in L^2.

The order $\leq$ on L now induces two natural orders on L^2:

- The *product order* $\leq$ is the point-wise extension of $\leq$, i.e., for all $x, y, x', y' \in L$,

$(x, y) \leq (x', y')$ iff $x \leq x'$ and $y \leq y'$;

- The *precision order* $\leq_p$ is defined as: for all $x, y, x', y' \in L, (x, y) \leq_p (x', y')$ iff $x \leq x'$ and $y' \leq y$.

The precision order can be motivated by the correspondence between pairs (x, y) and intervals $[x, y]$. Indeed, if $(x, y) \leq_p (x', y')$, then $[x, y] \supseteq [x', y']$. For our purposes, the precision order will be the most important one. It can easily be shown that both $\langle L^2, \leq \rangle$ and $\langle L^2, \leq_p \rangle$ are also lattices, which are complete iff the original lattice $\langle L, \leq \rangle$ is complete. Together with its two lattice orders, L^2 is called the *bilattice* corresponding to L.

Approximation theory is based on the study of operators on bilattices L^2 which are monotone w.r.t. the precision order $\leq_p$. Such operators are called *approximations*. For an approximation A and elements x, y of L, we denote by $A^1(x, y)$ and $A^2(x, y)$ the unique elements of L, for which $A(x, y) = (A^1(x, y), A^2(x, y))$. An approximation *approximates* an operator O on L if for each $x \in L$, $A(x, x)$ contains $O(x)$, i.e., $A^1(x, x) \leq O(x) \leq A^2(x, x)$. An *exact* approximation is one which maps exact elements to exact elements, i.e., $A^1(x, x) = A^2(x, x)$ for all $x \in L$. Similarly, a *consistent* approximation maps consistent elements to consistent elements, i.e., if $x \leq y$ then $A^1(x, y) \leq A^2(x, y)$. If an approximation is not consistent, it cannot approximate any operator. Each exact approximation is also consistent and approximates a unique operator O on L, namely that which maps each $x \in L$ to $A^1(x, x)$ (which is equal to $A^2(x, x)$). An approximation is *symmetric* if for each pair $(x, y) \in L^2$, if $A(x, y) = (x', y')$ then $A(y, x) = (y', x')$. Each symmetric approximation is also exact.

For an approximation A on L^2, the following two classes of operators on L can be defined: for each $y \in L$, the operator $A^1(\cdot, y)$ maps an element $x \in L$ to $A^1(x, y)$, i.e., $A^1(\cdot, y) = \lambda x.A^1(x, y)$; for each $x \in L$, the operator $A^2(x, \cdot)$ maps an element $y \in L$ to $A^2(x, y)$, i.e., $A^2(x, \cdot) = \lambda y.A^2(x, y)$. These operators are all monotone and, therefore, they each have a unique least fixpoint. We define an operator $C_A^{\downarrow}$ on L, which maps each $y \in L$ to $lfp(A^1(\cdot, y))$ and, similarly, an operator $C_A^{\uparrow}$, which maps each $x \in L$ to $lfp(A^2(x, \cdot))$. $C_A^{\downarrow}$ is called the *lower stable operator* of A, while $C_A^{\uparrow}$ is the *upper stable operator* of A. Both these operators are anti-monotone. Combining these two operators, the operator $\mathcal{C}_A$ on L^2 maps each pair (x, y) to $(C_A^{\downarrow}(y), C_A^{\uparrow}(x))$. This operator is called the *partial stable operator* of A. Because the lower and upper partial stable operators $C_A^{\downarrow}$ and $C_A^{\uparrow}$ are anti-monotone, the partial stable operator $\mathcal{C}_A$ is $\leq_p$-monotone. Note that if an approximation A is symmetric, its lower and upper partial stable operators will always be equal, i.e., $C_A^{\downarrow} = C_A^{\uparrow}$.

An approximation A defines a number of different fixpoints: the least fixpoint of an approximation A is called its *Kripke-Kleene fixpoint*, fixpoints of its partial stable operator $\mathcal{C}_A$ are *stable fixpoints* and the least fixpoint of $\mathcal{C}_A$ is called the *well-founded fixpoint* of A. As shown in (Denecker, Marek, and Truszczyński 2000) and (Denecker, Marek, and Truszczyński 2003), these fixpoints correspond to various semantics of logic programming, autoepistemic logic and default logic.

Finally, it should be noted that the concept of an approximation as defined in (Denecker, Marek, and Truszczyński 2000) corresponds to our definition of a *symmetric*

approximation.

2.2 Logic programming

In this section, we show how the formalism of logic programming and several of its extensions fit into the approximation theory framework. An *alphabet* or *vocabulary* Σ consists of a set Σ^o of object symbols, a set Σ^f of function symbols, and a set Σ^p of predicate symbols. Note that we make no formal distinction between variables and constants; both are simply called object symbols. There is no real difference here with the standard definitions, apart from the fact that this allows us to also place assignments of domain elements to "free variables" inside our interpretations, which means we do not have to introduce separate objects for this. We will still use the term "variables" to refer to those object symbols over which we quantify and call the other object symbols "constants". Predicate symbols start with a capital letter, function symbols are entirely lowercase and for object symbols we adopt the notational convention that variables are lowercase, whereas constants start with a capital.

As usual, a *term* is inductively defined as either an object symbol or an expression of the form $F(t_1, \ldots, t_n)$, where F/n is a function symbol and all the t_i are terms. An *atom* is of the form $P(t_1, \ldots, t_n)$, where P/n is a predicate symbol and all the t_i are terms. A *first-order logic formula* is inductively defined as:

- an atom is a formula;
- if φ is a formula then so is $\neg\varphi$;
- if φ is a formula and x an object symbol, then $\exists x\ \varphi$ is a formula;
- if φ and ψ are formulas, then so is $\varphi \vee \psi$.

As usual, a conjunction $\varphi \wedge \psi$ is defined as an abbreviation for $\neg(\neg\varphi \vee \neg\psi)$ and $\forall x\ \varphi$ abbreviates $\neg\exists x\ \neg\varphi$.

We will consider a quite general logic programming style language, which we call *rule sets*. A rule set Δ consists of rules of the form:

$$\forall \mathbf{x}\ P(\mathbf{t}) \leftarrow \varphi.$$

Here, P is a predicate symbol, $\mathbf{x}$ a tuple of variables, $\mathbf{t}$ a tuple of terms, and φ a first-order logic formula. For a rule r of the above form, the atom $P(\mathbf{t})$ is called the ***head*** of r, while φ is its ***body***. We denote these two parts of a rule r as, respectively, $head(r)$ and $body(r)$. Predicates that appear in the head of a rule are ***defined by*** Δ; all other predicates are ***open***. We denote the set of defined predicates by $Def(\Delta)$ and that of all open ones by $Op(\Delta)$.

We now define a class of semantics for such rule sets. We interpret an alphabet Σ by a Σ-*structure* or Σ-*interpretation*; such a Σ-interpretation I consists of a domain $dom(I)$, an interpretation of the object symbols Σ^o of Σ by domain elements, an interpretation of each function symbol $F/n \in \Sigma^f$ by an n-ary function on $dom(I)$, and an interpretation of each predicate symbol $P/n \in \Sigma^p$ by an n-ary relation on $dom(I)$.

A *pre-interpretation* of Σ consists of a domain and an interpretation of the object and function symbols $\Sigma^o \cup \Sigma^f$. If the alphabet Σ is clear from the context, we often omit this from our notation. For any symbol $\sigma \in \Sigma$, we denote by σ^I the interpretation of σ by I. Similarly, for a term t we denote the interpretation of t by t^I and we also extend this notation to tuples $\mathbf{t}$ of terms. For an interpretation I, an object symbol x, and a $d \in dom(I)$, we denote by $I[x/d]$ the interpretation with the same domain as I, that interprets x by d and coincides with I on all other symbols. We also extend this notation to tuples $\mathbf{x}$ and $\mathbf{d}$. We define an order $\leq$ on Σ-interpretations as: for all I, J, $I \leq J$ iff I and J share the same pre-interpretation and, for each predicate $P \in \Sigma^p$, $P^I \subseteq P^J$.

In logic programming, the domain is often restricted to the *Herbrand universe*, i.e., the set of all ground terms that can be constructed using the constants and function symbols in the alphabet. A *Herbrand pre-interpretation* is a pre-interpretation that has the Herbrand universe as its domain and interprets each constant and function symbol by itself. A *Herbrand interpretation* is an interpretation that extends some Herbrand pre-interpretation.

Let us recall that the standard satisfaction relation $I \models \varphi$ of first-order logic is inductively defined as:

- $I \models P(\mathbf{t})$ iff $\mathbf{t}^I \in P^I$;
- $I \models \varphi \vee \psi$ iff $I \models \varphi$ or $I \models \psi$;
- $I \models \neg\varphi$ iff $I \not\models \varphi$;
- $I \models \exists x\, \varphi$ iff there exists a $d \in dom(I)$ for which $I \models \varphi[x/d]$.

From now on, for a formula $\varphi(\mathbf{x})$ and a tuple $\mathbf{d}$ of domain elements, we freely use the more standard notation $I \models \varphi[\mathbf{x}/\mathbf{d}]$ or $I \models \varphi(\mathbf{d})$ instead of $I[\mathbf{x}/\mathbf{d}]$.

A feature of the stable and well-founded semantics for logic programs is that positive and negative occurrences of atoms in rule bodies are treated differently. The following truth evaluation function in *pairs* of interpretations captures this difference.

Definition 2.1. Let φ be a formula. Let I and J be interpretations that extend the same pre-interpretation. We now define when a formula φ is satisfied in the pair (I, J), denoted $(I, J) \models \varphi$, by induction over the size of φ:

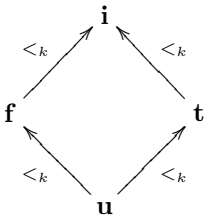
- $(I, J) \models P(\mathbf{t})$ iff $I \models P(\mathbf{t})$, i.e., $\mathbf{t}^I \in P^{\mathbf{I}}$;
- $(I, J) \models \neg\varphi$ iff $(J, I) \not\models \varphi$;
- $(I, J) \models \varphi \vee \psi$ iff $(I, J) \models \varphi$ or $(I, J) \models \psi$;
- $(I, J) \models \exists x\, \varphi$ iff there is a $d \in dom(I)$, such that $(I, J) \models \varphi[x/d]$.

Observe that evaluating the negation connective $\neg$ switches the roles of I and J. Because of this, positive occurrences of atoms are interpreted by I, while negative occurrences are interpreted by J. This satisfaction relation has a natural explanation when we view a pair $(I\ J)$ as an approximation, i.e., when I is seen as a lower estimate and J as an upper estimate of some interpretation K. When $I \leq K \leq J$, the above

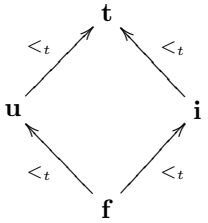
satisfaction relation underestimates the truth of positive occurrences of atoms—since it uses I for this—and the truth of negative occurrences of atoms is overestimated—here, it uses J. It follows that $(I, J) \models \varphi$ implies $K \models \varphi$, i.e., if $(I, J) \models \varphi$, then φ is certainly true in every approximated interpretation, while if $(I, J) \not\models \varphi$, then φ is possibly false. Vice versa, if we consider $(J, I) \models \varphi$, then positively occurring atoms are overestimated and negatively occurring ones are underestimated. Hence, if $(J, I) \not\models \varphi$, then φ is certainly false in all approximated interpretations, while if $(J, I) \models \varphi$, then φ is possibly true.

There is a strong duality between four-valued interpretations and pairs of two-valued interpretations, and also between three-valued interpretations and *consistent* pairs (I, J) of two-valued interpretations. Let us introduce a set $\mathcal{V}_3$ of truth values $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$, called *true*, *false* and *unknown*. We also define a set $\mathcal{V}_4$ of truth values $\{\mathbf{t}, \mathbf{f}, \mathbf{u}, \mathbf{i}\}$, where the additional truth value $\mathbf{i}$ is called *inconsistent*. We can define on $\mathcal{V}_4$ a *knowledge order* $\leq_k$ and a *truth order* $\leq_t$, as follows:

Knowledge order:



Truth order:



The structure $\langle \mathcal{V}_4, \leq_t, \leq_k \rangle$ is a complete bilattice, whereas $\mathcal{V}_3$ is a complete lattice with respect to the appropriate restriction of $\leq_t$, but is not even a lattice with respect to the restriction of $\leq_k$, since it does not contain $\mathit{lub}_{\leq_k}(\mathbf{t}, \mathbf{f}) = \top_k = \mathbf{i}$.

Let us now first redefine our concept of a standard two-valued interpretation, in such a way that it can easily be generalized to the three- or four-valued case. Let us fix a pre-interpretation F. An interpretation I that extends F now tells us, for each predicate P/n and each n-tuple of domain elements $\mathbf{d} \in dom(F)^n$, whether or not $\mathbf{d}$ belongs to the interpretation P^I of this predicate. We call such a pair $(P, \mathbf{d})$ a *domain atom* and also write it as $P(\mathbf{d})$. For a given set of predicates $\mathbf{P}$ and a pre-interpretation F, we denote the set of all domain atoms as $\mathcal{A}t^F_{\mathbf{P}}$. If F is a Herbrand pre-interpretation, then this set $\mathcal{A}t^F_{\mathbf{P}}$ of domain atoms is more commonly called the *Herbrand base*. Every interpretation I that extends a pre-interpretation F now corresponds in a unique way to a set of domain atoms or, equivalently, to a mapping ν from $\mathcal{A}t^F_{\Sigma^p}$ to the set of truth values $\mathcal{V}_2 = \{\mathbf{t}, \mathbf{f}\}$, where $\nu(P(\mathbf{d})) = \mathbf{t}$ means that the tuple $\mathbf{d}$ belongs to P^I and $\nu(P(\mathbf{d})) = \mathbf{f}$ means that it does not. We now define a *three-valued* interpretation as a mapping ν from $\mathcal{A}t^F_{\Sigma^p}$ to $\mathcal{V}_3$ and a *four-valued interpretation* as a mapping from $\mathcal{A}t^F_{\Sigma^p}$ to $\mathcal{V}_4$.

As mentioned above, three-valued interpretations correspond to consistent pairs of two-valued interpretations and four-valued interpretations correspond to arbitrary pairs. Let us denote by τ the isomorphism that maps each four-valued interpretation ν to the pair (I, J), where:

- I is such that $\mathbf{d} \in P^I$ iff $\nu(P(\mathbf{d}))$ is either $\mathbf{t}$ or $\mathbf{i}$;

- J is such that $\mathbf{d} \in P^J$ iff $\nu(P(\mathbf{d}))$ is either $\mathbf{t}$ or $\mathbf{u}$.

It is now easy to see that the restriction of τ to three-valued interpretations is indeed an isomorphism with the set of all consistent pairs (I, J).

The truth and knowledge orders defined on $\mathcal{V}_3$ and $\mathcal{V}_4$ induce obvious corresponding truth and knowledge orders on, respectively, three- and four-valued interpretations. Indeed, we can define $\nu \leq_t \nu'$ iff for each domain atom $P(\mathbf{d})$, $\nu(P(\mathbf{d})) \leq_t \nu'(P(\mathbf{d}))$, and similarly for $\leq_k$. Now, under the isomorphism τ, this truth order corresponds to the product order $\leq$ on pairs of interpretations, while this knowledge order corresponds to the precision order $\leq_p$, that is, for all ν and ν', with $\tau(\nu) = (I, J)$ and $\tau(\nu') = (I', J')$:

- $\nu \leq_t \nu'$ iff $(I, J) \leq (I', J')$ iff $I \leq I'$ and $J \leq J'$;
- $\nu \leq_k \nu'$ iff $(I, J) \leq_p (I', J')$ iff $I \leq I'$ and $J \geq J'$.

A three- or four-valued interpretation ν can be extended to a mapping of all sentences φ to, respectively, $\mathcal{V}_3$ or $\mathcal{V}_4$. This can be done in the following standard way. Let us first define the inverse $\cdot^{-1}$ of the truth values in $\mathcal{V}_4$ as follows:

$$\mathbf{t}^{-1} = \mathbf{f}; \qquad \mathbf{u}^{-1} = \mathbf{u};$$
$$\mathbf{f}^{-1} = \mathbf{t}; \qquad \mathbf{i}^{-1} = \mathbf{i}.$$

Let ν be a three- or four-valued interpretation, which extends some pre-interpretation F. We now inductively define the truth value of a formula φ according to ν, denoted by φ^ν, as:

- $(P(\mathbf{t}))^\nu = \nu(P(\mathbf{t}^F))$;
- $(\neg\varphi)^\nu = (\varphi^\nu)^{-1}$;
- $(\varphi \vee \psi)^\nu = \mathbf{v}$, where $\mathbf{v}$ is the *lub*$_{\leq_t}$ of the truth values φ^ν and ψ^ν;
- $(\exists x\ \varphi)^\nu = \mathbf{v}$, where $\mathbf{v}$ is the *lub*$_{\leq_t}$ of all truth values $(\varphi[x/d])^\nu$, for which $d \in dom(F)$.

These definitions correspond to Belnap's four-valued logic (Belnap 1977).

We now have the following correspondence between φ^ν and the satisfaction relation $\models$ that we defined above for pairs of interpretations. For consistent pairs, we get that if $\tau(\nu) = (I, J)$ then:

- $\varphi^\nu = \mathbf{t}$ iff $(I, J) \models \varphi$ and $(J, I) \models \varphi$;
- $\varphi^\nu = \mathbf{f}$ iff $(I, J) \not\models \varphi$ and $(J, I) \not\models \varphi$;
- $\varphi^\nu = \mathbf{u}$ iff $(I, J) \not\models \varphi$ and $(J, I) \models \varphi$;

In the four-valued case, we also have:

- $\varphi^\nu = \mathbf{i}$ iff $(I, J) \models \varphi$ and $(J, I) \not\models \varphi$;

We will sometimes also write $\varphi^{(I,J)}$ to denote the truth value φ^{ν}, where ν is such that $\tau(\nu) = (I, J)$.

For a set of predicates $\mathbf{P}$, we will denote by $\mathcal{L}^F_{\mathbf{P}}$ the class of all two-valued $(\Sigma^o \cup \Sigma^f \cup \mathbf{P})$-structures that extend some fixed pre-interpretation F. It is easy to see that $\langle \mathcal{L}^F_{\mathbf{P}}, \leq \rangle$ is a complete lattice. Given a pair of interpretations for the open predicates (O_1, O_2) in $(\mathcal{L}^F_{Op(\Delta)})^2$, we now define an immediate consequence operator $\mathcal{T}^{(O_1,O_2)}_{\Delta}$ on $(\mathcal{L}^F_{Def(\Delta)})^2$, i.e., on pairs of interpretations of the defined predicates. The definition below is an alternative formalization of the standard four-valued immediate consequence operator (Fitting 2002). The idea behind this operator is that if we give it a pair (I, J) approximating some $I \leq K \leq J$, it will produce a new, more precise estimate (I', J'), where the new underestimate I' will be constructed using the underestimates $(I, J) \models body(r)$ of the truth of the bodies of rules $r \in \Delta$, whereas the new overestimate J' is made using the overestimates $(J, I) \models body(r)$.

Definition 2.2. Let Δ be a rule set and $O_1, O_2 \in \mathcal{L}^F_{Op(\Delta)}$. We define a function $U^{(O_1,O_2)}_{\Delta}$ from $(\mathcal{L}^F_{Def(\Delta)})^2$ to $\mathcal{L}^F_{Def(\Delta)}$ as mapping each (I, J) to the interpretation I', such that for each $P/n \in Def(\Delta)$ and $\mathbf{d} \in dom(F)^n$, $\mathbf{d} \in P^{I'}$ iff there exists a rule $(\forall \mathbf{x}\, P(\mathbf{t}) \leftarrow \varphi(\mathbf{x})) \in \Delta$ and an $\mathbf{c} \in dom(F)^n$, such that $((O_1 \cup I), (O_2 \cup J)) \models \varphi(\mathbf{c})$ and $\mathbf{t}^{F[\mathbf{x}/\mathbf{c}]} = \mathbf{d}$. We define the operator $\mathcal{T}^{(O_1,O_2)}_{\Delta}$ on $(\mathcal{L}^F_{Def(\Delta)})^2$ as $\mathcal{T}^{(O_1,O_2)}_{\Delta}(I, J) = \big(U^{(O_1,O_2)}_{\Delta}(I, J), U^{(O_2,O_1)}_{\Delta}(J, I)\big)$.

When a pair (O_1, O_2) approximates an interpretation O for the open predicates, i.e., $O_1 \leq O \leq O_2$, then $\mathcal{T}^{(O_1,O_2)}_{\Delta}$ is an approximation of the well-known 2-valued immediate consequence operator T^O_{Δ}, which can be defined as $T^O_{\Delta}(I) = J$, with $(J, J) = \mathcal{T}^{(O,O)}_{\Delta}(I, I)$. Because $\mathcal{T}^{(O_1,O_2)}_{\Delta}$ is an approximation, it has a stable operator $\mathcal{C}_{\mathcal{T}^{(O_1\ O_2)}_{\Delta}}$. The *well-founded model* of Δ given (O_1, O_2) is the least fixpoint of this stable operator. Similarly, a pair (I, J) is a *partial stable model* of Δ given (O_1, O_2) iff (I, J) is a fixpoint of this stable operator. An interpretation I for which (I, I) is a partial stable model is called an (exact) stable model. Our language of rule sets subsumes that of normal logic programs. In particular, a normal logic program does not have any open predicates. In this case, $\mathcal{T}_{\Delta}$ is symmetric, which implies that its upper and lower stable operator coincide. Moreover, it turns out that this $C^{\uparrow}_{\mathcal{T}_{\Delta}} = C^{\downarrow}_{\mathcal{T}_{\Delta}}$ is identical to the well-known Gelfond-Lifschitz operator $\mathcal{GL}_{\Delta}$ (Gelfond and Lifschitz 1988). For the sake of completeness, we can also define some less popular logic programming semantics in terms of these operators. An interpretation I is a *supported model* of Δ given O iff I is a fixpoint of T^O_{Δ}. If Δ does not contain open predicates, then its supported models are known to coincide with the classical models of Clark's *completion* (Clark 1978). The *Kripke-Kleene model* (Fitting 1985) of Δ under an interpretation (O_1, O_2) for the open predicates is the pair (I, J) for which (I, J) is the least fixpoint of $\mathcal{T}^{(O_1,O_2)}_{\Delta}$.

A rule set Δ is *monotone* if every $\mathcal{T}^{(O_1\ O_2)}_{\Delta}$ is a monotone operator w.r.t. the product order $\leq$. For such rule sets, the well-founded model of Δ given some (O_1, O_2) can be shown to coincide with the Kripke-Kleene model of Δ under (O_1, O_2), which is also the unique partial stable model for Δ given (O_1, O_2). A rule set Δ is *positive* if no

defined predicate appears negatively in a rule body of Δ. Such rule sets are always monotone.

The following definition introduces some notation for the models of a rule set under partial stable and well-founded semantics. For an interpretation I for alphabet Σ and a subalphabet $\Sigma' \subseteq \Sigma$, we write $I|_{\Sigma'}$ to denote the restriction of I to Σ', i.e., $I|_{\Sigma'}$ is the Σ'-interpretation I' such that, for all $\sigma \in \Sigma'$, $\sigma^I = \sigma^{I'}$; similarly, for a pair (I, J) of Σ-interpretations, we write $(I, J)|_{\Sigma'}$ to denote the pair of Σ'-interpretations $(I|_{\Sigma'}, J|_{\Sigma'})$.

Definition 2.3. Let Δ be a rule set, F a pre-interpretation. and I, J structures in $\mathcal{L}^F_{\Sigma^p}$, i.e., they interpret all predicates of Δ. The pair (I, J) is a model of Δ under the well-founded semantics, denoted $(I, J) \models_w \Delta$ iff $(I, J)|_{Def(\Delta)}$ is the well-founded model of Δ given $(I, J)|_{Op(\Delta)}$. Similarly, (I, J) is a model of Δ under the partial stable model semantics, denoted $(I, J) \models_s \Delta$ iff $(I, J)|_{Def(\Delta)}$ is a partial stable model of Δ under $(I, J)|_{Op(\Delta)}$.

Using the above definitions, we can now characterize stable and well-founded semantics of the following extensions of logic programming:

- Normal logic programming: the bodies of rules are restricted to conjunctions of literals, there are no open predicates, and the pre-interpretation F is fixed to the Herbrand pre-interpretation.

- Abductive logic programming: the same, except that open predicates are allowed, which in this context are called *abducible* predicates and whose interpretation is arbitrary. LP-functions (Gelfond and Przymusinska 1996) are also of this form.

- Deductive Databases, and its extension AFP (Van Gelder 1993): intensional predicates are defined, extensional database predicates are open but interpreted by the database O.

- ID-logic (Denecker and Ternovska 2004): rule sets are used to represent inductive definitions.

All the results that we will prove for rule sets therefore apply to each of these formalisms. In the next chapters of this text, we will pay special attention to the language of ID-logic. We therefore now explain this in a bit more detail.

2.2.1 ID-logic

ID-logic (Denecker and Ternovska 2004) is an extension of classical first-order logic with a new construct for representing inductive definitions. Concretely, an inductive definition is represented by a rule set, enclosed by curly brackets $\{\}$. An ID-logic *formula* is a boolean combination of such inductive definitions and of first-order formulas. For instance, the following formula states that the undirected graph represented by the

predicates $Edge/2$ and $Node/1$ is connected:

$$\left\{\begin{array}{l}\forall x, y\ Path(x,y) \leftarrow Edge(x,y).\\ \forall x, z\ Path(x,z) \leftarrow \exists y\ Path(x,y) \wedge Path(y,z).\end{array}\right\}$$
$$\wedge\, \forall x, y\ Node(x) \wedge Node(y) \Rightarrow Path(x,y).$$

The semantics of ID-logic is defined by extending the usual inductive definition of the satisfaction relation with an additional base case to cover the new inductive definition primitive. Concretely, an inductive definition Δ will be interpreted according to the well-founded semantics for rule sets, i.e., for an interpretation I and ID-logic formulas φ, we define $I \models \varphi$ as:

- For an atom $P(\mathbf{t})$, $I \models P(\mathbf{t})$ iff $\mathbf{t}^I \in P^I$;
- For a definition Δ, $I \models \Delta$ iff $(I, I) \models_w \Delta$;
- For a conjunction $I \models \psi \wedge \varphi$, $I \models \psi \wedge \varphi$ iff $I \models \psi$ and $I \models \varphi$;
- And so on for the other connectives in the standard way.

We remark that, even though this definition uses the technical construct of the well-founded semantics, which can be three-valued, the eventual models of a definition are always two-valued. Given an interpretation O for the open predicates of a definition Δ, we call Δ *total* in O iff its well-founded model given (O, O) is exact, i.e., of the form (I, I). Obviously, the above definition implies that whenever $I \models \Delta$, then Δ is total in $I|_{Op(\Delta)}$.

2.3 Autoepistemic logic

In this section, we describe the syntax of autoepistemic logic and give a brief overview, based on (Denecker, Marek, and Truszczyński 2003), of how a number of different semantics for this logic can be defined using concepts from approximation theory.

Syntax and Semantics

Let $\mathcal{L}$ be the language of propositional logic based on a set of atoms Σ. Extending this language with a modal operator K, gives a language $\mathcal{L}_K$ of modal propositional logic. An autoepistemic theory is a set of formulas in this language $\mathcal{L}_K$. For such a formula φ, the subset of Σ containing all atoms which appear in φ, is denoted by $At(\varphi)$; atoms which appear in φ at least once outside the scope of the modal operator K are called *objective* atoms of φ and the set of all objective atoms of φ is denoted by $At_O(\varphi)$. A *modal literal* is a formula of the form $K(\psi)$, with ψ a formula. If φ is a subformula of ψ and φ appears negatively in ψ, we write $\varphi \in^- \psi$; if φ appears positively in ψ, we write $\varphi \in^+ \psi$. By the *K-rank* of an occurrence of a subformula φ in a formula ψ, we mean the number of modal operators in ψ, in whose scope φ occurs. As such, the objective atoms of ψ are precisely those atoms that have an occurrence of K-rank zero in ψ.

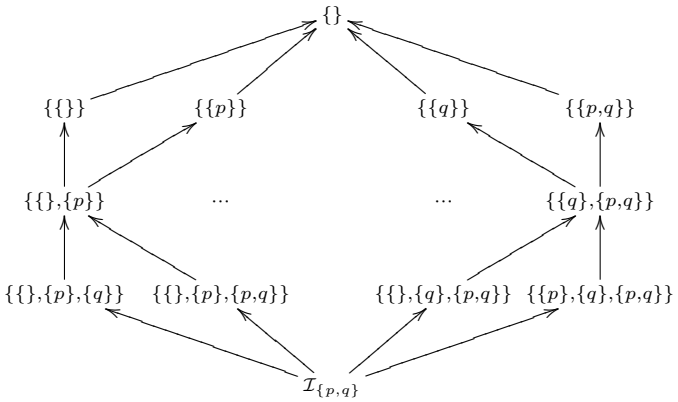


Figure 2.1: Part of the lattice $\mathcal{W}_{\{p,q\}}$.

To illustrate, consider the following example:

$$T = \{\varphi_1 = p \vee \neg Kp\,;\ \ \varphi_2 = K(p \vee Kq) \vee q\}$$

The objective atoms $At_O(\varphi_2)$ of φ_2 are $\{q\}$, while the atoms $At(\varphi_2)$ are $\{p, q\}$. The formula $K(p \vee Kq)$ is a modal literal of φ_2. We also have that $Kp \in^- \varphi_1$ and $p \in^+ \varphi_2$. The K-rank of q in $K(p \vee Kq)$ is 2, whereas the K-rank of the second occurrence of p in φ_1 is 1.

An *interpretation* or *world* is a subset of the alphabet Σ. The set of all interpretations of Σ is denoted by $\mathcal{I}_\Sigma$, i.e., $\mathcal{I}_\Sigma = 2^\Sigma$. A *possible world structure* is a set of interpretations, i.e. the set of all possible world structures $\mathcal{W}_\Sigma$ is defined as $2^{\mathcal{I}_\Sigma}$. Intuitively, a possible world structure sums up all "situations" which are possible. It therefore makes sense to order these according to inverse set inclusion to get a *knowledge order* $\leq$, i.e. for two possible world structures Q, Q', $Q \leq Q'$ iff $Q \supseteq Q'$. Indeed, if a possible world structure contains *more* possibilities, it actually contains *less* knowledge. Figure 2.1 shows part of the lattice $\mathcal{W}_{At(T)}$ for the above example T.

Following (Denecker, Marek, and Truszczyński 2003), we will define the semantics of an autoepistemic theory by an operator on the bilattice $\mathcal{B}_\Sigma = \mathcal{W}_\Sigma^2$. An element $(P\ S)$ of $\mathcal{B}_\Sigma$ is known as a *belief pair* and is called *consistent* iff $P \leq S$. In a consistent belief pair $(P\ S)$, P can be viewed as describing what must *certainly* be known, i.e., as giving an *under*estimate of what is known, while S can be viewed as denoting what might *possibly* be known, i.e. as giving an *over*estimate. Based on this intuition, there are two ways of estimating the truth of modal formulas according to $(P\ S)$. To conservatively estimate the truth of a formula φ in a world I and a consistent belief pair (P, S), we evaluate all positively occurring modal literals $K\psi \in^+ \varphi$ in the possible world set with the least knowledge, i.e., in P, and all negatively occurring modal literals in the possible world set with the most knowledge, i.e., in S. Vice versa, to liberally estimate the truth of a formula φ in I and (P, S), we evaluate positive modal literals in S and negative modal literals in P. To formalize these intuitions, we define the following

truth assignment:

Definition 2.4. For each $(P, S) \in \mathcal{B}_\Sigma$, $I \in \mathcal{I}_\Sigma$ and formula φ in alphabet Σ, we inductively define $\mathcal{H}_{I,(P,S)}(\varphi)$ as:

- For each atom p, $\mathcal{H}_{I,(P,S)}(p) = \mathbf{t}$ iff $p \in I$;
- $\mathcal{H}_{I,(P,S)}(\varphi_1 \wedge \varphi_2) = \mathbf{t}$, iff $\mathcal{H}_{I,(P,S)}(\varphi_1) = \mathbf{t}$ and $\mathcal{H}_{I,(P,S)}(\varphi_2) = \mathbf{t}$;
- $\mathcal{H}_{I,(P,S)}(\varphi_1 \vee \varphi_2) = \mathbf{t}$, iff $\mathcal{H}_{I,(P,S)}(\varphi_1) = \mathbf{t}$ or $\mathcal{H}_{I,(P,S)}(\varphi_2) = \mathbf{t}$;
- $\mathcal{H}_{I,(P,S)}(\neg\varphi) = \neg\mathcal{H}_{I,(S,P)}(\varphi)$;
- $\mathcal{H}_{I,(P,S)}(K\varphi) = \mathbf{t}$ iff $\mathcal{H}_{J,(P,S)}(\varphi) = \mathbf{t}$ for all $J \in P$;

This evaluation function has two important properties. First, if we consider an exact belief pair, i.e., one of the form (Q, Q), then $\mathcal{H}_{I,(Q,Q)}(\varphi)$ corresponds to the standard S_5 evaluation (Meyer and van der Hoek 1995) of φ in the possible world structure Q and world I. Second, there is an exact sense in which this function can be used to conservatively or liberally estimate the truth of a formula φ. A conservative estimate can be achieved by considering $\mathcal{H}_{I,(P,S)}(\varphi)$. It can then be shown that for any possible world structure Q, with $P \leq Q \leq S$, it is indeed the case that $\mathcal{H}_{I,(P,S)}(\varphi) \leq \mathcal{H}_{I,(Q,Q)}(\varphi)$. Conversely, a liberal estimate consists of $\mathcal{H}_{I,(S,P)}(\varphi)$ and, indeed, for any possible world structure Q, with $P \leq Q \leq S$, it is indeed the case that $\mathcal{H}_{I,(S,P)}(\varphi) \geq \mathcal{H}_{I,(Q,Q)}(\varphi)$.

We remark that the evaluation $\mathcal{H}_{I,(P,S)}(K\varphi)$ of a modal literal $K\varphi$ depends only on (P, S) and not on I. We will sometimes emphasize such properties by replacing the irrelevant symbol by a dot, e.g., by writing $\mathcal{H}_{\cdot,(P,S)}(K\varphi)$. Similarly, $\mathcal{H}_{I,(P,S)}(\varphi)$ of an objective formula φ depends only on I and we also write $\mathcal{H}_{I,(\cdot,\cdot)}(\varphi)$.

The conservative and liberal way of estimating the truth of a theory can now be used to derive a new, more precise belief pair (P', S') from an original pair (P, S). First, we will focus on constructing the new overestimate S'. As S' needs to overestimate knowledge, it needs to contain as few interpretations as possible. This means that S' should consist of only those interpretations, which manage to satisfy the theory even if the truth of its modal literals is conservatively estimated. So, S' should contains those interpretations I for which, for all φ in T, $\mathcal{H}_{I,(P,S)}(\varphi) = \mathbf{t}$. Conversely, to construct the new underestimate P', we need as many interpretations as possible. This means that P' should contain those interpretations I which satisfy the theory, when liberally evaluating its modal literals, i.e., for which, for all φ in T, $\mathcal{H}_{I,(S,P)}(\varphi) = \mathbf{t}$.

These intuitions motivate the following definition of the operator $\mathcal{D}_T$ on $\mathcal{B}_\Sigma$:

$$\mathcal{D}_T(P, S) = (\mathcal{D}^u_T(S, P), \mathcal{D}^u_T(P, S))$$

with $\mathcal{D}^u_T(P, S) = \{I \in \mathcal{I}_\Sigma \mid \forall\varphi \in T : \mathcal{H}_{I,(P,S)}(\varphi) = \mathbf{t}\}$.

It can be illuminating to reformulate this definition using more standard concepts and notation. Given a pair (P, S) and a formula φ, it is obviously the case that, in any evaluation $\mathcal{H}_{I,(P,S)}(\varphi)$, all *positively* occurring modal literals $K\psi$ will be interpreted as $\mathcal{H}_{\cdot,(P,S)}(K\psi)$, while all *negatively* occurring modal literals $K\psi$ will be interpreted as $\mathcal{H}_{\cdot,(S,P)}(K\psi)$. Let us denote by $\varphi\langle P, S\rangle$ the formula φ' that is the result of filling

in these truth values, i.e., of replacing each top-level modal literal by $\mathbf{t}$ or $\mathbf{f}$ in the appropriate way. Every such $\varphi\langle P, S\rangle$ is of course simply a propositional formula. What the function $\mathcal{D}^u_T$ now actually does is simply map a pair (P, S) to the set $Mod(T\langle P, S\rangle)$ of all classical models of the propositional theory $T\langle P, S\rangle = \{\varphi\langle P, S\rangle \mid \varphi \in T\}$. So, the operator $\mathcal{D}_T$ can be equivalently defined as:

$$\mathcal{D}_T(P, S) = (Mod(T\langle S, P\rangle), Mod(T\langle P, S\rangle)).$$

It can be shown that every operator $\mathcal{D}_T$ is a symmetric approximation (Denecker, Marek, and Truszczyński 2003), which therefore approximates a unique operator on $\mathcal{W}_\Sigma$, namely the operator D_T which maps each Q to $\mathcal{D}^u_T(Q, Q)$. This operator D_T is precisely the operator considered in (Moore 1984). As shown in (Denecker, Marek, and Truszczyński 2003), these operators define a family of semantics for a theory T:

- fixpoints of D_T are *expansions* of T (Moore 1984),
- fixpoints of $\mathcal{D}_T$ are *partial expansions* of T (Denecker, Marek, and Truszczynski 1998),
- the least fixpoint of $\mathcal{D}_T$ is the *Kripke-Kleene fixpoint* of T (Denecker, Marek, and Truszczynski 1998),
- fixpoints of $C^{\downarrow}_{\mathcal{D}_T}$ are *extensions* of T (Denecker, Marek, and Truszczyński 2003),
- fixpoints of $\mathcal{C}_{\mathcal{D}_T}$ are *partial extensions* of T (Denecker, Marek, and Truszczyński 2003)
- the least fixpoint of $\mathcal{C}_{\mathcal{D}_T}$ is the *well-founded model* of T (Denecker, Marek, and Truszczyński 2003).

These various dialects of autoepistemic logic differ in their treatment of "ungrounded" expansions (Konolige 1988), i.e., expansions arising from cyclicities such as $Kp \Rightarrow p$.

Example 2.1. To illustrate these definitions, we will compute the Kripke-Kleene model of our example theory $T = \{p \vee \neg Kp;\ K(p \vee Kq) \vee q\}$. This computation starts at the least precise element $(\mathcal{I}_{\{p,q\}}, \{\})$ of $\mathcal{B}_{\{p,q\}}$. We first construct the new underestimate $\mathcal{D}^u_T(\{\}, \mathcal{I}_{\{p,q\}}) = Mod(T\langle\{\}, \mathcal{I}_{\{p,q\}}\rangle)$. It is easy to see that, for the negatively occurring modal literal Kp,

$$\mathcal{H}_{\cdot,(\mathcal{I}_{\{p,q\}},\{\})}(Kp) = \mathbf{f},$$

and for the positively occurring modal literal $K(p \vee Kq)$,

$$\mathcal{H}_{\cdot,(\{\},\mathcal{I}_{\{p,q\}})}(K(p \vee Kq)) = \mathbf{t}.$$

Therefore, $T\langle\{\}, \mathcal{I}_{\{p,q\}}\rangle = \{p \vee \neg\mathbf{f};\ q \vee \mathbf{t}\} = \{\mathbf{t}\}$ and $\mathcal{D}^u_T(\{\}, \mathcal{I}_{\{p,q\}}) = \mathcal{I}_{\{p,q\}}$. Now, to compute the new overestimate $\mathcal{D}^u_T(\mathcal{I}_{\{p,q\}}, \{\}) = Mod(T\langle\mathcal{I}_{\{p,q\}}, \{\}\rangle)$, we note that

$$\mathcal{H}_{\cdot,(\{\},\cdot)}(Kp) = \mathbf{t},$$

and

$$\mathcal{H}_{\cdot,(\mathcal{I}_{\{p,q\}},\cdot)}(K(p \vee Kq)) = \mathbf{f}.$$

Therefore, $T\langle\mathcal{I}_{\{p,q\}},\{\}\rangle = \{p \vee \neg\mathbf{t}; q \vee \mathbf{f}\} = \{p; q\}$ and $\mathcal{D}^u_T(\mathcal{I}_{\{p,q\},\{\}}) = \{\{p,q\}\}$. So, $\mathcal{D}_T(\mathcal{I}_{\{p,q\}},\{\}) = (\mathcal{I}_{\{p,q\}},\{\{p,q\}\})$.

To compute $\mathcal{D}^u_T(\{\{p,q\}\},\mathcal{I}_{\{p,q\}})$, we note that it is still the case that:

$$\mathcal{H}_{\cdot,(\mathcal{I}_{\{p,q\}},\cdot)}(Kp) = \mathbf{f} \text{ and } \mathcal{H}_{\cdot,(\{\{p,q\}\},\cdot)}(K(p \vee Kq)) = \mathbf{t}.$$

So, $\mathcal{D}^u_T(\{\{p,q\}\},\mathcal{I}_{\{p,q\}}) = \mathcal{I}_{\{p,q\}}$. Similarly, still both $\mathcal{H}_{\cdot,(\{\{p,q\}\},\cdot)}(Kp) = \mathbf{t}$ and $\mathcal{H}_{\cdot,(\mathcal{I}_{\{p,q\}},\cdot)}(K(p \vee Kq)) = \mathbf{f}$. So, $\mathcal{D}^u_T(\mathcal{I}_{\{p,q\}},\{\{p,q\}\}) = \{\{p,q\}\}$. Therefore, $(\mathcal{I}_{\{p,q\}},\{\{p,q\}\})$ is the least fixpoint of $\mathcal{D}_T$, i.e., the Kripke-Kleene model of T.

2.4 Default logic

Let $\mathcal{L}$ be the language of propositional logic for an alphabet Σ. A *default* d is a formula

$$\frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma}$$

with $\alpha, \beta_1, \ldots, \beta_n, \gamma$ formulas of $\mathcal{L}$. The formula γ is called the consequence $cons(d)$ of d. A *default theory* is a pair $\langle D, W\rangle$, with D a set of defaults and W a set of formulas of $\mathcal{L}$.

(Konolige 1987) suggested a transformation m from default logic to autoepistemic logic, which was shown by (Denecker, Marek, and Truszczyński 2003) to capture the semantics of default logic. For simplicity, we will ignore the original formulation of the semantics of default logic and view this as being defined by the autoepistemic theory $m(\langle D, W\rangle)$.

Definition 2.5. Let $\langle D, W\rangle$ be a default theory and let $d = \frac{\alpha:\beta_1,\ldots,\beta_n}{\gamma}$ be a default in D. Then

$$m(d) = (K\alpha \wedge \neg K\neg\beta_1 \wedge \cdots \wedge \neg K\neg\beta_n \Rightarrow \gamma)$$

and

$$m(\langle D, W\rangle) = \{m(d) \mid d \in D\} \cup W.$$

A pair (P, S) of possible world structures is an expansion (Marek and Truszczyński 1989), a partial expansion (Denecker, Marek, and Truszczyński 2003), an extension (Reiter 1980), a partial extension (Denecker, Marek, and Truszczyński 2003), the Kripke-Kleene model (Denecker, Marek, and Truszczyński 2003) or the well-founded model (Baral and Subrahmanian 1991) of a default theory $\langle D, W\rangle$ if it is, respectively, an expansion, a partial expansion, an extension, a partial extension, the Kripke-Kleene or the well-founded model of $m(D, W)$. The semantics of extensions is the most common.

Chapter 3

Modularity results

3.1 Introduction

An important aspect of human reasoning is that it is often incremental in nature. When dealing with a complex domain, we tend to initially restrict ourselves to a small subset of all relevant concepts. Once these "basic" concepts have been figured out, we then build another, more "advanced", layer of concepts on this knowledge. A quite illustrative example of this can be found in most textbooks on computer networking. These typically present a seven-layered model of the way in which computers communicate. First, in the so-called physical layer, there is only talk of hardware and concepts such as wires, cables and electronic pulses. Once these low-level issues have been dealt with, the resulting knowledge becomes a *fixed* base, upon which a new layer, the data-link layer, is built. This no longer considers wires and cables and so on, but rather talks about packages of information travelling from one computer to another. Once again, after the workings of this layer have been figured out, this information is "taken for granted" and becomes part of the foundation upon which a new layer is built. This process continues all the way up to a seventh layer, the application layer, and together all of these layers describe the operation of the entire system.

In this chapter, we investigate a formal equivalent of this method. More specifically, we address the question of whether a formal theory in some non-monotonic language can be split into a number of different *levels* or *strata*, such that the formal semantics of the entire theory can be constructed by successively constructing the semantics of the various strata. We will use the terms "stratification" and "splitting" interchangeably to denote a division into a number of different levels. This is a more general use of both these terms, than in literature such as (Apt, Blair, and Walker 1988) and (Gelfond 1987). Such stratifications are interesting from both a theoretical, knowledge representational and a more practical point of view.

On the more theoretical side, stratification results provide crucial insight into the semantics of a language, and hence in its use for knowledge representation. Indeed, the human brain seems unsuited for holding large chunks of unstructured information. When the complexity of a domain increases, we rely on our ability to understand

and describe parts of the domain and construct a description of the whole domain by composing the descriptions of its components. Large theories which cannot be understood as somehow being a composition of components, simply cannot be understood by humans. Stratification results are, therefore, important, especially in the context of non-monotonic languages, where adding a new expression to a theory might affect the meaning of what was already represented. Our results will present cases where adding a new expression is guaranteed *not* to alter the meaning of the existing theory.

On the more practical side, computing models of a theory by incrementally constructing models of each of its levels might offer considerable computational gain. Indeed, suppose that, normally, it takes $t(n)$ time to construct the model(s) of a theory of size n. If we were able to split such a theory into, say, m smaller theories of equal size n/m, we could use this stratification to compute the model(s) of the theory in $m \cdot t(n/m)$ time. As model generation is typically quite hard, i.e., $t(n)$ is a large function of n, this could provide quite a substantial improvement. Of course, much depends of the value of m. Indeed, in the worst case, the theory would allow only the trivial stratification in which the entire theory is a single level, i.e., $m = 1$, which obviously does not lead to any gain. However, because, as argued above, human knowledge tends to exhibit a more modular structure, we would expect real knowledge bases to be rather well-behaved in this respect.

Because of these reasons, it is not surprising that stratifiability and related concepts, such as Dix's notion of modularity (Dix 1995), have already been intensively studied. Indeed, splitting results have been proven for autoepistemic logic under the semantics of expansions (Gelfond and Przymusinska 1992; Niemelä and Rintanen 1994) default logic under the semantics of extensions (Turner 1996) and various kinds of logic programs under the stable model semantics (Lifschitz and Turner 1994; Erdoğan and Lifschitz 2004; Eiter, Gottlob, and Mannila 1997). In all of these works, stratification is seen as a syntactical property of a theory in a certain language under a certain formal semantics.

Here, we will take the different, more general approach of studying this topic at the abstract level of approximation theory. Let us briefly sketch the method that we will follow. Approximation theory defines a family of different kinds of fixpoints of operators and shows that these correspond to a family of semantics for a number of different logics. We will introduce the concept of a *stratifiable operator* and prove that such operators can be split into a number of smaller *component operators*, in such a way that the different kinds of fixpoints of the original operator can be constructed by incrementally constructing the corresponding fixpoints of its component operators. These algebraic results will then be used to derive concrete splitting results for logic programming, autoepistemic logic and default logic. To do this, we will follow these two steps:

- First, we determine *syntactical* conditions which suffice to ensure that every operator corresponding to a theory, that satisfies these conditions, is in fact a stratifiable operator. This tells us that the models of such a theory under various semantics, i.e., the various kinds of fixpoints of the associated operator, can be constructed by incrementally constructing the corresponding fixpoints of the components of this operator.

- Second, we also need to provide a precise, computable characterization of the components of stratifiable operators. This will be done by presenting a *syntactical* method of deriving a number of smaller theories from the original theory and showing that the components of the original operator are precisely the operators associated with these new theories.

So, in other words, using the algebraic characterization of the semantics of a number of different logics by approximation theory, our algebraic results show how splitting can be done on a semantical level, and deriving concrete splitting results for a specific logic simply boils down to determining which syntactical notions correspond to our semantical splitting concepts.

3.2 Stratification in approximation theory

In this section, we develop a theory of stratifiable operators. We will, in section 3.2.2, investigate operators on a special kind of lattice, namely *product lattices*, which will be introduced in section 3.2.1. In section 3.2.3, we then return to approximation theory and discuss stratifiable approximations on product lattices. Finally, in Sections 3.2.4 and 3.2.5, we will introduce some additional concepts, which will be useful when applying our abstract results to a concrete logic.

3.2.1 Product lattices

We begin by defining the notion of a *product set*, which is a generalization of the well-known concept of Cartesian products.

Definition 3.1. Let I be a set, which we will call the *index set* of the product, and for each $i \in I$, let S_i be a set. The *product set* $\otimes_{i\in I} S_i$ is the following set of functions:

$$\otimes_{i\in I} S_i = \{f \mid f : I \to \bigcup_{i\in I} S_i \text{ such that } \forall i \in I : f(i) \in S_i\}.$$

Intuitively, a product set $\otimes_{i\in I} S_i$ contains all ways of selecting one element from each set S_i. As such, if the index set I is a set with n elements, e.g. the set $\{1, \ldots, n\}$, the product set $\otimes_{i\in I} S_i$ is simply (isomorphic to) the cartesian product $S_1 \times \cdots \times S_n$.

Definition 3.2. Let I be a set and for each $i \in I$, let $\langle S_i, \leq_i\rangle$ be a partially ordered set. The *product order* $\leq$ on the set $\otimes_{i\in I} S_i$ is defined as, for all $x, y \in \otimes_{i\in I} S_i$:

$$x \leq y \qquad iff \qquad \forall i \in I : x(i) \leq_i y(i).$$

It is easy to see that if all the $\langle S_i, \leq_i\rangle$ are (complete) lattices, then $\langle \otimes_{i\in I} S_i, \leq\rangle$ is also a (complete) lattice. We therefore refer to $\langle \otimes_{i\in I} S_i, \leq\rangle$ as the *product lattice* of the lattices S_i.

From now on, we will always assume a well-founded partial order $\preceq$ on the index set I. This will allow us to use inductive arguments in dealing with elements of product lattices. Most of our results, however, also hold for index sets with an arbitrary partial

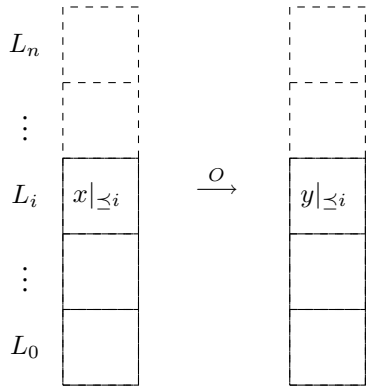


Figure 3.1: O is stratifiable if $x|_{\preceq i}$ determines $O(x)|_{\preceq i}$.

order; if a certain proof depends on the well-foundedness of I, we will always explicitly mention this.

For an element x of a product lattice $\otimes_{i\in I} L_i$ and $i \in I$, we abbreviate $x|_{\{j\in I | j\preceq i\}}$ by $x|_{\preceq i}$. We also use similar abbreviations $x|_{\prec i}$, $x|_i$ and $x|_{\not\preceq i}$. If i is a minimal element of the well-founded set I, $x|_{\prec i}$ is defined as the empty function. For each index i, the set $\{x|_{\preceq i} \mid x \in L\}$, ordered by the appropriate restriction $\leq|_{\preceq i}$ of the product order, is also a lattice. Clearly, this sublattice of L is isomorphic to the product lattice $\otimes_{j\preceq i} L_i$. We denote this sublattice by $L|_{\preceq i}$ and use a similar notation $L|_{\prec i}$ for $\otimes_{j\prec i} L_i$. For each element x of a product lattice L and each index $i \in I$, the extension $x|_{\prec i} \sqcup x|_i$ of $x|_{\prec i}$ is clearly equal to $x|_{\preceq i}$. For ease of notation, we sometimes simply write $x(i)$ instead of $x|_i$ in such expressions, i.e., we identify an element a of the ith lattice L_i with the function from $\{i\}$ to L_i which maps i to a. Similarly, $x|_{\prec i} \sqcup x(i) \sqcup x|_{\not\preceq i} = x$.

We will use the symbols x, y to denote elements of an entire product lattice L; a, b to denote elements of a single level L_i and u, v to denote elements of $L|_{\prec i}$.

3.2.2 Operators on product lattices

We want to consider operators on a lattice that is split into a number of different levels L_i. Formally, let $\langle I, \preceq\rangle$ be a well-founded index set and let $L = \otimes_{i\in I} L_i$ be the product lattice of lattices $\langle L_i, \leq_i\rangle_{i\in I}$. We now want to look at those operators O on L that respect this ordering of the components of L. An operator O on L will be called stratifiable over the order $\preceq$, if the value $(O(x))(i)$ of $O(x)$ in the ith stratum only depends on values $x(j)$ for which $j \preceq i$. This concept is illustrated in Figure 3.1.

Definition 3.3. An operator O on a product lattice L is *stratifiable* iff for all $x, y \in L$ and $i \in I$: if $x|_{\preceq i} = y|_{\preceq i}$ then $O(x)|_{\preceq i} = O(y)|_{\preceq i}$.

It is also possible to characterize stratifiability in a more constructive manner. The following theorem shows that stratifiability of an operator O on a product lattice L is

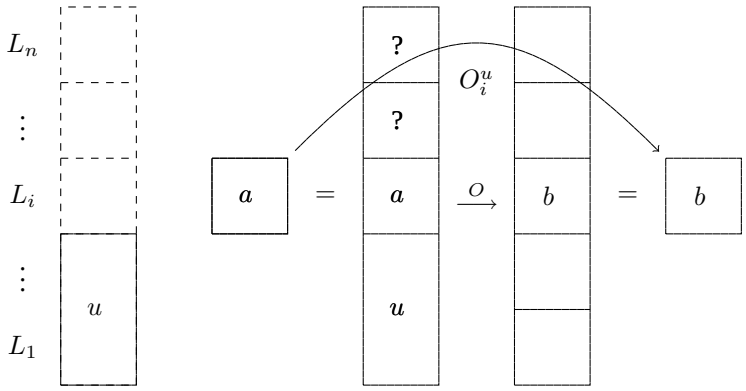


Figure 3.2: The component O_i^u of O maps a to $O(x)|_i$, with x any extension of $u \sqcup a$.

equivalent to the existence of a family of operators on each lattice L_i—one for each u in $L|_{\prec i}$—which mimics the behaviour of O on this lattice.

Proposition 3.1. *Let O be an operator on a product lattice L. O is stratifiable iff for each $i \in I$ and $u \in L|_{\prec i}$ there exists a unique operator O_i^u on L_i, such that for all $x \in L$:*

$$\textit{If } x|_{\prec i} = u \textit{ then } (O(x))(i) = O_i^u(x(i)).$$

Proof. To prove the implication from left to right, let O be a stratifiable operator, $i \in I$ and $u \in L|_{\prec i}$. We define the operator O_i^u on L_i as

$$O_i^u : L_i \to L_i : a \mapsto (O(y))(i),$$

with y some element of L extending $u \sqcup a$. Because of the stratifiability of O, this operator is well-defined and it trivially satisfies the required condition.

To prove the other direction, suppose the right-hand side of the equivalence holds and let x, x' be elements of L, such that $x|_{\preceq i} = x'|_{\preceq i}$. Then for each $j \preceq i$:

$$(O(x))(j) = O_j^{x|_{\prec j}}(x(j)) = O_j^{x'|_{\prec j}}(x'(j)) = (O(x'))(j).$$

□

These operators O_i^u will play an important role in our results. We will call them the *components* of O. The construction of these components is illustrates in Figure 3.2.

Definition 3.4. Let O be a stratifiable operator on a product lattice L with index set I and let $i \in I$ and $u \in L|_{\prec i}$. The *component* O_i^u of O on level i given u is the unique operator on $L|_i$ that satisfies the property stated in Proposition 3.1 or, equivalently, that maps each $a \in L|_i$ to $O(x)|_i$, where x is any element of L that extends both u and a.

These components of O can be used to construct the fixpoints of a stratifiable operator in a bottom-up manner w.r.t. the well-founded order $\preceq$ on the index set.

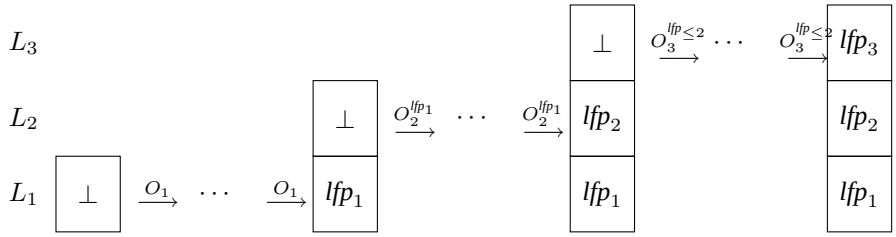


Figure 3.3: The components of O can be used to construct, for instance, $lfp(O)$.

Theorem 3.1. *Let O be a stratifiable operator on a product lattice L. Then for each $x \in L$:*

$$x \text{ is a fixpoint of } O \qquad iff \qquad \forall i \in I : x(i) \text{ is a fixpoint of } O_i^{x|_{\prec i}}.$$

Proof. Follows immediately from Proposition 3.1. □

If O is a monotone operator on a complete lattice, we are often interested in its *least* fixpoint. This can also be constructed by means of the least fixpoints of the components of O. Figure 3.3 illustrates how this could be done. However, such a construction of course requires each component to actually have a least fixpoint as well. We will therefore first show that the components of a monotone operator are also monotone.

Proposition 3.2. *Let O be a stratifiable operator on a product lattice L, which is monotone w.r.t. the product order $\leq$. Then for each $i \in I$ and $u \in L|_{\prec i}$, the component $O_i^u : L_i \to L_i$ is monotone w.r.t. to the order $\leq_i$ of the ith lattice L_i of L.*

Proof. Let i be an index in I, u an element of $L|_{\prec i}$ and a, b elements of L_i, such that $a \leq_i b$. Let $x, y \in L$, such that x extends $u \sqcup a$, y extends $u \sqcup b$ and for each $j \not\preceq i$, $x(j) = y(j)$. Because of the definition of $\leq$, clearly $x \leq y$. Therefore, for all $j \in I$:

$$O_j^{x|_{\prec j}}(x(j)) = (O(x))(j) \leq_j (O(y))(j) = O_j^{y|_{\prec j}}(y(j)).$$

Taking $j = i$, this now implies $O_i^u(a) \leq_i O_i^u(b)$. □

Now, we can prove that the least fixpoints of the components of a monotone stratifiable operator indeed form the least fixpoint of the operator itself. We will do this, by first proving the following, slightly more general theorem, which we will be able to reuse later on.

Proposition 3.3. *Let O be a monotone operator on a complete product lattice L and let for each $i \in I$, $u \in L|_{\prec i}$, P_i^u be a monotone operator on L_i (not necessarily a component of O), such that:*

$$x \text{ is a fixpoint of } O \qquad iff \qquad \forall i \in I : x(i) \text{ is a fixpoint of } P_i^{x|_{\prec i}}.$$

Then the following equivalence also holds:

$$x \text{ is the least fixpoint of } O \qquad iff \qquad \forall i \in I : x(i) \text{ is the least fixpoint of } P_i^{x|_{\prec i}}.$$

Proof. To prove the implication from left to right, let x be the least fixpoint of O and let i be an arbitrary index in I. We will show that for each fixpoint a of $P_i^{x|_{\prec i}}$, $a \geq x(i)$. So, let a be such a fixpoint. We can inductively extend $x|_{\prec i} \sqcup a$ to an element y of L by defining for all $j \not\preceq i$, $y(j)$ as $lfp(P_j^{y|_{\prec j}})$. Because of the well-foundedness of $\preceq$, y is well defined. Furthermore, y is clearly also a fixpoint of O. Therefore $x \leq y$ and, by definition of the product order on L, $x(i) \leq_i y(i) = a$.

To prove the other direction, let x be an element of L, such that, for each $i \in I$, $x(i)$ is the least fixpoint of $P_i^{x|_{\prec i}}$. Now, let y be the least fixpoint of O. To prove that $x = y$, it suffices to show that for each $i \in I$, $x|_{\preceq i} = y|_{\preceq i}$. We will prove this by by induction on the well-founded order $\preceq$ of I. If i is a minimal element of I, the proposition trivially holds. Now, let i be an index which is not the minimal element of I and assume that for each $j \prec i$, $x|_{\preceq j} = y|_{\preceq j}$. It suffices to show that $x(i) = y(i)$. Because y is a fixpoint of O, $y(i)$ is fixpoint of $P_i^{y|_{\prec i}}$. As the induction hypothesis implies that $x|_{\prec i} = y|_{\prec i}$, $y(i)$ is a also fixpoint of $P_i^{x|_{\prec i}}$ and therefore $x(i) \leq y(i)$. However, because x is also a fixpoint of O and therefore must be greater than the least fixpoint y of O, the definition of the product order on L implies that $x(i) \geq y(i)$ as well. Therefore $x(i) = y(i)$. □

It is worth noting that the condition that the order $\preceq$ on I should be well-founded is necessary for this proposition to hold. Indeed, this can be demonstrated by the following example.

Example 3.1. Let L be the product lattice $\otimes_{i\in\mathbb{Z}} L_i$, with $\mathbb{Z}$ the integers ordered by their usual, non-well-founded order and each L_i the lattice $\{0, 1\}$ ordered by $0 \leq 1$. Let O be the operator that maps each $x \in L$ to the following element $y \in L$:

$$y : \mathbb{Z} \to \{0, 1\} : \begin{cases} i \mapsto 0 & \text{if } x(i-1) = 0; \\ i \mapsto 1 & \text{otherwize.} \end{cases}$$

This O is monotone and, therefore, it has a least fixpoint, which turns out to be the bottom element $\bot$ of L, which maps each $i \in \mathbb{Z}$ to 0. Also, O is stratifiable over the order $\leq$ of $\mathbb{Z}$. Its components can be characterized as follows. For every $i \in \mathbb{Z}$ and $u \in L|_{<i}$, if $u(i-1) = 0$, then the component O_i^u is the constant operator that maps both 0 and 1 to 0; otherwize O_i^u is the constant operator that maps 0 and 1 to 1. We now know that an $x \in L$ is a fixpoint of O iff for each i, $x(i)$ is a fixpoint of $O_i^{x|_{<i}}$. However, it is not the case that if, for each i, $x(i)$ is the *least* fixpoint of $O_i^{x|_{<i}}$, then also x is the least fixpoint of O. Indeed, for the element $\top$ that maps each i to 1, each of the operators $O_i^{\top|_{<i}}$ has $\top(i) = 1$ as its least fixpoint, but nevertheless $\top$ is not equal to the least fixpoint $\bot$ of O.

Together with Theorem 3.1 and Proposition 3.2, this Proposition 3.3 of course implies that for each stratifiable operator O on a product lattice L, an element $x \in L$ is the least fixpoint of O iff $\forall i \in I$, $x(i)$ is the least fixpoint of $O_i^{x|_{\prec i}}$. In other words, the least fixpoint of a stratifiable operator can also be incrementally constructed.

3.2.3 Approximations on product lattices

In section 2.1.2, we introduced several concepts from approximation theory, pointing out that we are mainly interested in studying Kripke-Kleene, stable and well-founded fixpoints of approximations. Similar to our treatment of arbitrary operators in the previous section, we will in this section investigate the relation between these various fixpoints of an approximation and its components. In doing so, it will be convenient to switch to an alternative representation of the bilattice L^2 of a product lattice $L = \otimes_{i \in I} L_i$. Indeed, this bilattice is clearly isomorphic to the structure $\otimes_{i \in I} L_i^2$, i.e., to a product lattice of bilattices. From now on, we will not distinguish between these two representations. More specifically, when viewing A as a stratifiable operator, it will be convenient to consider its domain equal to $\otimes_{i \in I} L_i^2$, but when viewing A as an approximation, the representation $(\otimes_{i \in I} L_i)^2$ is more natural.

Obviously, this isomorphism and the results of the previous section already provide a way of constructing the Kripke-Kleene fixpoint of a stratifiable approximation A, by means of its components A_i^u. Also, it is clear that if A is both exact and stratifiable, then the unique operator O approximated by A is stratifiable as well. Indeed, this is a trivial consequence of the fact that $A(x, x) = (O(x), O(x))$ for each $x \in L$.

These results leave only the stable and well-founded fixpoints of A to be investigated. We will first examine the operators $A^1(\cdot, y)$ and $A^2(x, \cdot)$, and then move on to the lower and upper stable operators $C_A^{\downarrow}$ and $C_A^{\uparrow}$, before finally getting to the partial stable operator $\mathcal{C}_A$ itself.

Proposition 3.4. *Let L be a product lattice and let $A : L^2 \to L^2$ be a stratifiable approximation. Then, for each $x, y \in L$, the operators $A^1(\cdot, y)$ and $A^2(x, \cdot)$ are also stratifiable. Moreover, for each $i \in I$ and $u \in L|_{\prec i}$, the components of these operators are:*

$$(A^1(\cdot, y))_i^u = (A_i^{(u, y|_{\prec i})})^1(\cdot, y(i));$$
$$(A^2(x, \cdot))_i^u = (A_i^{(x|_{\prec i}, u)})^2(x(i), \cdot).$$

Proof. Let x, y be elements of L, i an element of I. Then, because A is stratifiable, $(A(x, y))(i) = (A_i^{(x,y)|_{\prec i}})(x(i), y(i))$. From this, the two equalities follow. □

In the previous section, we showed that the components of a monotone operator are monotone as well (Proposition 3.2). This result obviously implies that the components A_i^u of a stratifiable approximation are also approximations. Therefore, such a component A_i^u also has lower and upper stable operators $C_{A_i^u}^{\downarrow}$ and $C_{A_i^u}^{\uparrow}$. It turns out that the lower and upper stable operators of the components of A characterize the components of the lower and upper stable operators of A.

Proposition 3.5. *Let L be a product lattice and let A be a stratifiable approximation on L^2. Then the operators $C_A^{\downarrow}$ and $C_A^{\uparrow}$ are also stratifiable. Moreover, for each $x, y \in L$,*

$$x = C_A^{\downarrow}(y) \quad \textit{iff} \quad \textit{for each } i \in I, x(i) = C^{\downarrow}_{A_i^{(x,y)|_{\prec i}}}(y(i));$$
$$y = C_A^{\uparrow}(x) \quad \textit{iff} \quad \textit{for each } i \in I, y(i) = C^{\uparrow}_{A_i^{(x,y)|_{\prec i}}}(x(i)).$$

Proof. Let x, y be elements of L. By Proposition 3.4, $x = C^{\downarrow}_A(y) = lfp(A^1(\cdot, y))$ iff for each $i \in I$:

$$x(i) = lfp\big((A^1(\cdot, y))^{x|_{\prec i}}_i\big) = lfp\big((A^{(x,y)|_{\prec i}}_i)^1(\cdot, y(i))\big) = C^{\downarrow}_{A^{(x,y)|_{\prec i}}_i}(y(i)).$$

The proof of the second equivalence is analogous. □

This proposition shows how, for each $x, y \in L$, $C^{\downarrow}_A(y)$ and $C^{\uparrow}_A(x)$ can be constructed incrementally from the upper and lower stable operators corresponding to the components of A. This result also implies a similar property for the partial stable operator $\mathcal{C}_A$ of an approximation A.

Proposition 3.6. *Let L be a product lattice and let $A : L^2 \to L^2$ be a stratifiable approximation. Then the operator $\mathcal{C}_A$ is also stratifiable. Moreover, for each $x, x', y, y' \in L$, the following equivalence holds:*

$$(x', y') = \mathcal{C}_A(x, y) \qquad iff \qquad \forall i \in I : \begin{cases} x'(i) = C^{\downarrow}_{A^{(x',y)|_{\prec i}}_i}(y(i)); \\ y'(i) = C^{\uparrow}_{A^{(x,y')|_{\prec i}}_i}(x(i)). \end{cases}$$

Proof. The above equivalence follows immediately from Proposition 3.5. To prove the stratifiability of $\mathcal{C}_A$, let $x_1, y_1, x_2, y_2 \in L$, such that $(x_1, y_1)|_{\preceq i} = (x_2, y_2)|_{\preceq i}$. Let $(x'_1, y'_1) = \mathcal{C}_A(x_1, y_1)$ and $(x'_2, y'_2) = \mathcal{C}_A(x_2, y_2)$. It suffices to show that $\forall j \preceq i$, $x'_1(j) = x'_2(j)$ and $y'_1(j) = y'_2(j)$. We show this by induction over $\preceq$. First, if j is minimal, it follows from $y_1(j) = y_2(j)$ and $x_1(j) = x_2(j)$ that $C^{\downarrow}_{A_j}(y_1(j)) = C^{\downarrow}_{A_j}(y_2(j))$ and $C^{\uparrow}_{A_j}(x_1(j)) = C^{\uparrow}_{A_j}(x_2(j))$. Second, if j is not minimal, then $C^{\downarrow}_{A^{(x'_1,y_1)|_{\prec j}}_j}(y_1(j)) = C^{\downarrow}_{A^{(x'_2,y_2)|_{\prec j}}_j}(y_2(j))$ and $C^{\uparrow}_{A^{(x_1,y'_1)|_{\prec j}}_j}(x_1(j)) = C^{\uparrow}_{A^{(x_2,y'_2)|_{\prec j}}_j}(x_2(j))$, because $y_1|_{\preceq j} = y_2|_{\preceq j}$ and $x_1|_{\preceq j} = x_2|_{\preceq j}$, while the induction hypothesis also states that $x'_1|_{\prec j} = x'_2|_{\prec j}$ and $y'_1|_{\prec j} = y'_2|_{\prec j}$. □

It should be noted that the components $(\mathcal{C}_A)^{(u,v)}_i$ of the partial stable operator of a stratifiable approximation A are, in general, not equal to the partial stable operators $\mathcal{C}_{A^{(u,v)}_i}$ of the components of A. Indeed, $(\mathcal{C}_A)^{(u,v)}_i = ((C^{\downarrow}_A)^v_i, (C^{\uparrow}_A)^u_i))$, whereas $\mathcal{C}_{A^{(u,v)}_i} = (C^{\downarrow}_{A^{(u,v)}_i}, C^{\uparrow}_{A^{(u,v)}_i})$. Clearly, these two pairs are not necessarily equal, as $(C^{\downarrow}_A)^v_i$ ignores the argument u, which does appear in $C^{\downarrow}_{A^{(u,v)}_i}$. We can, however, characterize the fixpoints of $\mathcal{C}_A$, i.e., the partial stable fixpoints of A, by means of the partial stable fixpoints of the components of A.

Theorem 3.2. *Let L be a product lattice and let $A : L^2 \to L^2$ be a stratifiable approximation. Then for each element (x, y) of L^2:*

$$(x, y) \textit{ is a fixpoint of } \mathcal{C}_A \qquad iff \qquad \forall i \in I : (x, y)|_i \textit{ is a fixpoint of } \mathcal{C}_{A^{(x,y)|_{\prec i}}_i}.$$

Proof. Let x, y be elements of L, such that $(x, y) = \mathcal{C}_A(x, y)$. By Proposition 3.6, this is equivalent to for each $i \in I$, $x = C^{\downarrow}_{A^{(x,y)|_{\prec i}}_i}(y(i))$ and $y = C^{\uparrow}_{A^{(x,y)|_{\prec i}}_i}(x(i))$. □

By Proposition 3.3, this theorem has the following corollary:

Corollary 1. *Let L be a product lattice and let $A : L^2 \to L^2$ be a stratifiable approximation. Then for each element (x, y) of L^2:*

$$(x, y) = lfp(\mathcal{C}_A) \qquad \textit{iff} \qquad \forall i \in I : (x, y)|_i = lfp(\mathcal{C}_{A_i^{(x,y)|_{\prec i}}}).$$

Putting all of this together, the main results of this section can be summarized as follows. If A is a stratifiable approximation on a product lattice L, then a pair (x, y) is a fixpoint, Kripke-Kleene fixpoint, stable fixpoint, or well-founded fixpoint of A iff for each $i \in I$, $(x, y)|_i$ is, respectively, a fixpoint, Kripke-Kleene fixpoint, stable fixpoint, or well-founded fixpoint of the component $A_i^{(x,y)|_{\prec i}}$ of A. Moreover, if A is exact then an element $x \in L$ is a fixpoint of the unique operator O approximated by A iff for each $i \in I$, $(x(i), x(i))$ is a fixpoint of the component $A_i^{(x,x)|_{\prec i}}$ of A. These characterizations give us a way of incrementally constructing each of these fixpoints.

3.2.4 Dependency relations

The previous sections have studied operators and approximations that are stratifiable over some product lattice. Therefore, to apply these results to some particular operator $O : L \to L$, we first need to come up with an appropriate way of dividing L into a number of different levels L_i. In this section, we develop a uniform way of doing this, starting from a detailed analysis of the structure of the operator. As a starting point, we will again assume that L is isomorphic to some product lattice $\otimes_{i \in I} L_i$. However, now, we do not assume an *a priori* order on the index set I and we will also not work towards stratifying the operator O over this particular product lattice. Instead, the product $\otimes_{i \in I} L_i$ is meant to capture some structure that is evident in the operator O. For instance, if O operates on the lattice of all interpretations of some propositional alphabet, then each L_i could be the interpretation of a single atom of this alphabet. From this initial, fine-grained product $\otimes_{i \in I} L_i$, we will now derive a different, coarser product lattice $\otimes_{j \in J} L'_j$, isomorphic to it, over which we *can* stratify O.

The main idea is to examine the internal structure of O w.r.t. the component lattices L_i of L. For instance, what information about x is used by O to determine the value $(O(x))(i)$ of $O(x)$ in a component lattice L_i? Does such an $(O(x))(i)$ depend on the value $x(j)$ of x in each L_j? Or is there some $J \subset I$, such that the restriction $x|_J$ of x to this J already completely determines what $(O(x))(i)$ will be? The following concept captures these basic dependencies expressed by an operator. For a binary relation θ on a set S and $y \in S$, we write (θy) for $\{x \in S \mid x \theta y\}$.

Definition 3.5. Let O be an operator on a lattice $L = \otimes_{i \in I} L_i$. A binary relation $\rightsquigarrow$ on I is a *dependency relation* of O iff for all $i \in I$ and $x, y \in L$, if $x|_{(\rightsquigarrow i)} = y|_{(\rightsquigarrow i)}$, then $(O(x))(i) = (O(y))(i)$.

An operator can have many dependency relations. In fact, any superset of a dependency relation is also a dependency relation. Therefore, smaller dependency relations are more informative. However, an operator does not necessarily have a least dependency relation. This can be shown by the following example.

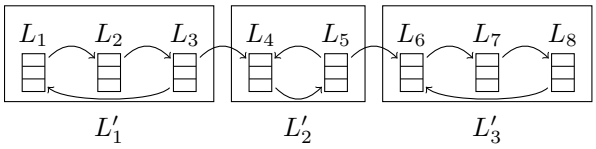


Figure 3.4: Given a dependency relation for an operator O on $\otimes L_i$, we can stratify O over $\otimes L'_j$.

Example 3.2. We consider the product lattice $\otimes_{n\in\mathbb{N}} L_n$ with each $L_n = \{0, 1\}$. Let O be the operator on this lattice defined as $O(x) = y$, with for all $n > 0$, $y(n) = 0$ and $y(0) = 1$ iff there exists $n \in \mathbb{N}$ such that for all $m \geq n$, $x(m) = 0$. For each n, let $\rightsquigarrow_n$ be the binary relation consisting of the tuples $m \rightsquigarrow_n 0$ for which $m \geq n$. Now, each $\rightsquigarrow_n$ is a dependency relation of O. As such, a least dependency relation $\rightsquigarrow_\perp$ of O would have to be some subset of the intersection $\cap_{n\in\mathbb{N}} \rightsquigarrow_n$ of all these relations $\rightsquigarrow_n$. However, $\cap_{n\in\mathbb{N}} \rightsquigarrow_n = \{\}$, so $\rightsquigarrow_\perp$ would also have to be $\{\}$, but this is not a dependency relation of O.

Given a dependency relation $\rightsquigarrow$ for an operator O on $L = \otimes_{i\in I} L_i$, we can now proceed to restructure L into a different product lattice, such that O is stratifiable with respect to this new structure. For this, we need to consider the reflexive, transitive closure of $\rightsquigarrow$, which we will denote as $\leq_\rightsquigarrow$. The following definition is illustrated in Figure 3.4.

Definition 3.6. Let O be an operator on a product lattice $\otimes_{i\in I} L_i$. Let $\langle J, \preceq\rangle$ be a well-founded poset. A partition $(I_j)_{j\in J}$ of I *respects* $\rightsquigarrow$ iff for all $i \in I_j$ and $i' \in I_{j'}$, if $i \leq_\rightsquigarrow i'$, then $j \preceq j'$.

Let us remark that, for any such partition $(I_j)_{j\in J}$, it must be the case that for every $(\leq_\rightsquigarrow)$-equivalence class $\bar{\imath} = \{i' \in I \mid i \leq_\rightsquigarrow i' \text{ and } i' \leq_\rightsquigarrow i\}$, there exists a unique $j \in J$ such that $\bar{\imath} \subseteq I_j$. In fact, if we consider the set $\overline{I}$ of all such equivalence classes, together with the obvious order $\preceq_\rightsquigarrow$ on $\overline{I}$ that is induced by $\leq_\rightsquigarrow$, then it is easy to see that the partition $(\bar{\imath})_{\bar{\imath}\in\overline{I}}$ itself respects $\rightsquigarrow$. Moreover, this will be the most fine-grained of all such partitions. As such, a dependency relation directly provides a constructive way of deriving the most detailed partition that respects it. It can also easily be seen that the order $\preceq_\rightsquigarrow$ is well-founded iff $\rightsquigarrow$ is.

Clearly, for any partition $(I_j)_{j\in J}$ of I, the original product $L = \otimes_{i\in I} L_i$ is isomorphic to $\otimes_{j\in J} \otimes_{i\in I_j} L_j$. We can therefore also view O as an operator on the product lattice $\otimes_{j\in J} L'_j$, where each L'_j is $\otimes_{i\in I_j} L_j$. It is now obvious that, on this product lattice, O is stratifiable.

Proposition 3.7. *Let O be an operator on $L = \otimes_{i\in I} L_i$ and let $\rightsquigarrow$ be a dependency relation of O. For all partitions $(I_j)_{j\in J}$ of I that respect $\rightsquigarrow$, O is stratifiable on $\otimes_{j\in J} L|_{I_j}$.*

Therefore, if we want to stratify some operator O on $\otimes_{i\in I} L_i$, it suffices to find a

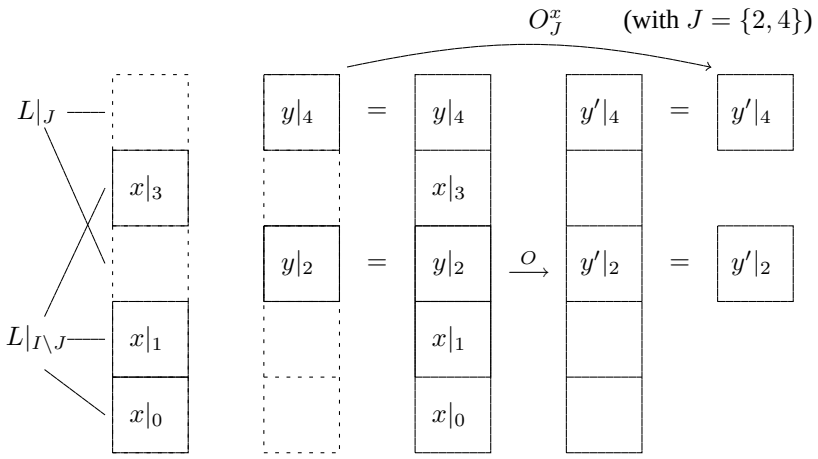


Figure 3.5: The recombination O^x_J maps $y \in L|_J$ to $O(x \sqcup y)|_J$.

well-founded dependency relation for O, as this will give us a way of constructing a product lattice on which we can stratify O.

3.2.5 Recombinations

While the main purpose of our stratification results is to break up a big operator O into smaller component operators, we will sometimes also be interested in putting some of these components back together to form a new set of operators. We now make this more precise. Let O be a stratifiable operator on a product lattice $\otimes_{i\in I} L_i$. For a subset J of I and $x \in L|_{I\setminus J}$, we denote by O^x_J the operator on $L|_J$ which maps each $y \in L|_J$ to $O(x \sqcup y)|_J$. Such operators O^x_J are called *recombinations of* O. This concept is illustrated in Figure 3.5.

Our goal is now to show that, for each partition $\mathcal{J}$ of I, a stratifiable operator O can be split into the recombinations O^x_J, with $J \in \mathcal{J}$. Let us remark that because O is stratifiable, we of course already know that this must be the case for the trivial partition $(\{i\})_{i\in I}$. We will prove the desired result by showing that a recombination O^x_J is also stratifiable and can be split into the components of O itself.

Proposition 3.8. *Let O be a stratifiable operator. For each $J \subseteq I$ and $x \in L|_{I\setminus J}$, O^x_J is stratifiable.*

Proof. Let O^x_J be as above, $i \in J$, and $y, y' \in L|_J$, such that $y|_{\preceq i} = y'|_{\preceq i}$. By definition, $O^x_J(y) = O(x \sqcup y)|_J$. Because $(x \sqcup y)|_{\preceq i} = (x \sqcup y')|_{\preceq i}$, we have that, by stratifiability of O,

$$O^x_J(y)|_{\preceq i} = O(x \sqcup y)|_{\{j\in J | j\preceq i\}} = O(x \sqcup y')|_{\{j\in J | j\preceq i\}} = O^x_J(y')|_{\preceq i}.$$

□

Proposition 3.9. *Let O be a stratifiable operator. For each $J \subseteq I$, $x \in L|_{I \setminus J}$, $i \in J$, and $u \in L|_{\{j \in J | j \prec i\}}$, the component $(O^x_J)^u_i$ of O^x_J is equal to the component $O_i^{u \sqcup (x|_{\prec i})}$ of O.*

Proof. Let $(O^x_J)^u_i$ be as above and let $y \in L_i$. By definition, for any z extending $u \sqcup y$ to J, $(O^x_J)^u_i(y) = O(x \sqcup z)|_i = (O_i^{(x \sqcup z)|_{\prec i}}(z|_i))|_i = O_i^{x|_{\prec i} \sqcup u}(y)$. □

These two propositions now imply the wanted result.

Theorem 3.3. *Let O be a stratifiable operator and let $\mathcal{J}$ be a partition of I. Then, for each $x \in L$, x is a fixpoint (the least fixpoint, a stable fixpoint, or the well-founded fixpoint) of O (assuming that O is monotone or an approximation, where appropriate) iff for each $J \in \mathcal{J}$, $x|_J$ is a fixpoint (respectively, the least fixpoint, a stable fixpoint, or the well-founded fixpoint) of $O_J^{x|_{I \setminus J}}$.*

Proof. We only show the correspondence between fixpoints; the proofs of the other correspondences are similar. Let x be a fixpoint of O. By Theorem 3.1, this is equivalent to: $\forall i \in I$, $x|_i$ is a fixpoint of $O_i^{x|_{\prec i}}$. Because $\mathcal{J}$ partitions I, this is equivalent to $\forall J \in \mathcal{J}$, $\forall i \in J$, $x|_i$ is a fixpoint of $O_i^{x|_{\prec i}}$. By Proposition 3.9, such a component $O_i^{x|_{\prec i}}$ of O is equal to the component $(O_J^{x|_{I \setminus J}})_i^{x|_{\{j \in J | j \prec i\}}}$ of the recombination $O_J^{x|_{I \setminus J}}$. By Proposition 3.8 and Theorem 3.1, $\forall J \in \mathcal{J}$, $\forall i \in J$, $x|_i$ is a fixpoint $(O_J^{x|_{I \setminus J}})_i^{x|_{\{j \in J | j \prec i\}}}$ iff $\forall J \in \mathcal{J}$, $x|_J$ is a fixpoint of $O_J^{x|_{I \setminus J}}$. □

3.3 Application to logic programming

In this section, we will apply our algebraic stratification results to logic programming. We will first start by considering a simplified language and then show how these results extend to the general rule sets considered in Section 2.2.

3.3.1 The propositional case

For this section, we consider propositional logic programs without open predicates. Let Σ be a propositional alphabet, i.e., a set of propositional atoms, denoted as $P, Q, \ldots$ A logic program in alphabet Σ is a set of rules $P \leftarrow \varphi$, with $P \in \Sigma$ and φ a propositional formula of Σ. Because propositional logic programs are simply a special case of the general rule sets considered in Section 2.2, we have already defined a number of different semantics for such programs. We recall that, for a rule set Δ, these definitions made use of the operator $\mathcal{T}_\Delta$ on pairs of interpretations of the defined predicates of Δ. In our current setting, we therefore need to consider the set of all interpretations of the alphabet Σ, which is simply isomorphic to power set 2^Σ of Σ. We denote the lattice $\langle 2^\Sigma, \subseteq \rangle$ as $\mathcal{I}_\Sigma$. We will use the symbol Π to refer to propositional logic programs and reserve the symbol Δ for the general case of arbitrary rule sets.

Our algebraic results made use of the concept of a dependency relation for an operator on a product lattice. We recall that, for an operator O on a product lattice $\otimes_{i \in I} L_i$, a binary relation on the index set I is called a dependency relation of O iff

it is the case that whenever, for some $i \in I$ and $x, x' \in \otimes_{i \in I} L_i$, $x|_{(\rightsquigarrow i)} = x'|_{(\rightsquigarrow i)}$, then $(O(x))|_i = (O(x'))|_i$. For logic programs, we are interested in the operator $\mathcal{T}_\Pi$ on $\mathcal{I}^2_\Sigma$. Let us now take as an index set I the alphabet Σ itself. For each atom $P \in \Sigma$, we then need a lattice L_P. For this, we will use the lattice $\{\mathbf{t}, \mathbf{f}\}$ of truth values that can be assigned to this one atom P. It is now clear that $\mathcal{I}_\Sigma$ is isomorphic to the product $\otimes_{P \in \Sigma} L_P$. Given this isomorphism, we now see that a dependency relation for the operator $\mathcal{T}_\Pi$ is a binary relation $\rightsquigarrow$ on the alphabet Σ, such that whenever, for some atom P and pairs $(I, J), (I', J') \in \mathcal{I}^2_\Sigma$, $(I, J)|_{(\rightsquigarrow P)} = (I', J')|_{(\rightsquigarrow P)}$, then $(\mathcal{T}_\Pi(I, J))|_P = (\mathcal{T}_\Pi(I', J'))|_P$.

We now consider a class of binary relations, which can be defined using only the program Π itself, such that every relation in this class is in fact a dependency relation of the operator $\mathcal{T}_\Pi$. Intuitively, the idea behind this definition is that an atom P depends on an atom Q if for some rule $P \leftarrow \varphi$ of Π, Q affects the truth of φ.

Definition 3.7. Let Π be a logic program in alphabet Σ. Let $\rightsquigarrow$ be a binary relation on Σ. We call $\rightsquigarrow$ a *dependency relation for* Π if the following condition is satisfied: whenever two pairs of interpretations (I, J) and (I', J') coincide on all atoms $Q \in \Sigma$ for which $Q \rightsquigarrow P$, then for all rules of Π of the form $P \leftarrow \varphi$, $(I, J) \models \varphi$ iff $(I', J') \models \varphi$.

It follows immediately from the definition of $\mathcal{T}_\Pi$ that any such dependency relation for Π is indeed a dependency relation for this operator. As such, our results from Section 3.2.4 show that $\mathcal{T}_\Pi$ is stratifiable over any partition of its alphabet that respects such a dependency relation.

Definition 3.8. Let Π be a logic program in alphabet Σ and let $\langle I, \preceq \rangle$ be a well-founded poset. A partition $(\Sigma_i)_{i \in I}$ of Σ is called a *splitting* of Π if it respects one of the dependency relations of Π.

To illustrate this definition, we consider the following program:

$$E = \left\{ \begin{array}{l} P \leftarrow \neg Q, \neg R. \\ Q \leftarrow \neg P, \neg R. \\ S \leftarrow P, Q. \end{array} \right\}.$$

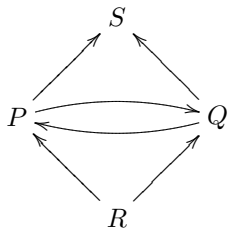
We now construct a dependency relation for this. In general, for any program Π, there is an obvious way to construct one of its dependency relations.

Definition 3.9. Let Π be a logic program with alphabet Σ. The *obvious dependency relation* $\rightsquigarrow_\Pi$ for Π is defined as, for all $P, Q \in \Sigma$:

$$Q \rightsquigarrow_\Pi P \qquad iff \qquad \exists r \in \Pi, \text{ such that } Q \text{ appears in } body(r) \text{ and } P = head(r).$$

It can easily be seen that every obvious dependency relation $\rightsquigarrow_\Pi$ is indeed a dependency relation for Π.

The obvious dependency relation of the example E is the following:



Using this dependency relation, it can easily be seen that the following partition of Σ is a splitting of E: we use the totally ordered set $\{0, 1, 2\}$ as index set and define $\Sigma_0 = \{R\}$, $\Sigma_1 = \{P, Q\}$ and $\Sigma_2 = \{S\}$. Therefore, the operator $\mathcal{T}_E$ for this example is stratifiable on the product lattice $\mathcal{I}_{\Sigma_0} \times \mathcal{I}_{\Sigma_1} \times \mathcal{I}_{\Sigma_2}$.

In general, if we have splitting $(\Sigma_i)_{i\in I}$ for a program Π, then the operator $\mathcal{T}_\Pi$ is stratifiable on the product lattice $\otimes_{i\in I}\mathcal{I}_{\Sigma_i}$, which, by Theorem 3.2, implies that we can split all of the operators $T_\Pi, \mathcal{T}_\Pi$ and $\mathcal{GL}_\Pi$. In other words, it is possible to split logic programs w.r.t. the supported model, Kripke-Kleene, stable model and well-founded semantics. Moreover, the supported, Kripke-Kleene, stable and well-founded models of Π can be computed from, respectively, the supported, Kripke-Kleene, stable and well-founded models of the components of the operator $\mathcal{T}_\Pi$.

In order to be able to perform this construction in practice, however, we also need a more constructive characterization of these components. We will now show how to derive new logic programs from the original program, such that these components correspond to an operator associated to these new programs. First, we will define the restriction of a program to a subset of its alphabet.

Definition 3.10. Let Π be a logic program with a splitting $(\Sigma_i)_{i\in I}$. For each $i \in I$, the program Π_i consists of all clauses which have an atom from Σ_i in their head.

In the case of our example, the program E is partitioned in $\{E_0, E_1, E_2\}$ with $E_0 = \{\}$, $E_1 = \{P \leftarrow \neg Q, \neg R; Q \leftarrow \neg P, \neg R\}$ and $E_2 = \{S \leftarrow P, Q\}$.

If Π has a splitting $(\Sigma_i)_{i\in I}$, then any program Π_i contains only atoms from $\bigcup_{j\preceq i}\Sigma_j$. Given a pair (U, V) of interpretations of $\bigcup_{j\prec i}\Sigma_j$, we can therefore construct a program containing only atoms from Σ_i by replacing all other atoms by their truth-value according to (U, V).

Definition 3.11. Let Π be a logic program with a splitting $(\Sigma_i)_{i\in I}$. For each $i \in I$ and $(U, V) \in B_\Sigma|_{\prec i}$, we define $\Pi_i\langle U, V\rangle$ as the new logic program Π', which results from replacing each literal l whose atom is in $\bigcup_{j\prec i}\Sigma_j$ by its truth value in (U, V), i.e., a positive literal P is replaced by $\mathbf{t}$ if $P \in U$ (and by $\mathbf{f}$ otherwise), whereas a negative literal $\neg P$ is replaced by $\mathbf{t}$ if $P \notin V$ (and by $\mathbf{f}$ otherwise),

It is now easy to see that the programs constructed in this way are now precisely those which characterize the components of the operator $\mathcal{T}_\Pi$.

Theorem 3.4. *Let Π be a logic program with a splitting $(\Sigma_i)_{i\in I}$. For each $i \in I$, $(U, V) \in B_\Sigma|_{\prec i}$ and $(A, B) \in B_{\Sigma_i}$:*

$$(\mathcal{T}_\Pi)_i^{(U,V)}(A, B) = (U_{\Pi_i\langle U,V\rangle}(A, B), U_{\Pi_i\langle V,U\rangle}(B, A)).$$

It is worth noting that this theorem implies that a component $(\mathcal{T}_\Pi)_i^{(U,V)}$ is, in contrast to the operator $\mathcal{T}_\Pi$ itself, not necessarily exact, i.e., we might need different programs for constructing the under- and overestimates.

As a side remark, let us recall that in Section 2.2, we considered an isomorphism τ between pairs of two-valued interpretations and four-valued interpretations. This isomorphism allows to also consider the operator $\mathcal{T}_\Pi$ as an operator on four-valued interpretations, i.e., we can identify $\mathcal{T}_\Pi$ with the operator $\tau^{-1} \circ \mathcal{T}_\Pi \circ \tau$ that maps each four-valued interpretation ν to ν' for which $\nu' = \tau^{-1}(\mathcal{T}_\Pi(\tau(\nu)))$. In this setting, we could also have formulated the above definition in the following equivalent way. For a four-valued interpretations ν, we denote by $\Pi_i\langle\nu\rangle_4$ the result of replacing each literal l whose atom is in $\bigcup_{j\prec i}\Sigma_j$ by its four-valued truth value l^ν. The component $(\mathcal{T}_\Pi)_i^\mu$ is then $\mathcal{T}_{\Pi_i\langle\mu\rangle_4}$. So, instead of characterizing such a component using two programs that contain two-valued truth-values, we can equivalently characterize it using a single program that contains four-valued truth values.

We will call a pair (I, J) of interpretations the *stratified well-founded model* of Π with respect to a splitting $(\Sigma_i)_{i\in I}$ iff for each $i \in I$, $(I, J)|_i$ is the well-founded fixpoint of the component $(\mathcal{T}_\Pi)_i^{(I,J)|_{\prec i}}$. Given the above remark, this is equivalent to $(I, J) = \tau(\nu)$, where ν is the four-valued interpretation such that, for each $i \in I$, $\nu|_i$ is the well-founded model of the program $\Pi_i\langle\mu\rangle_4$, with $\mu = \nu|_{\prec j}$. Similarly, we say that (I, J) is a *stratified partial stable model* or the *stratified Kripke-Kleene model* of Π with respect to the splitting $(\Sigma_i)_{i\in I}$ iff for each $i \in I$, $(I, J)|_i$ is, respectively, a stable fixpoint or the least fixpoint of the component $(\mathcal{T}_\Pi)_i^{(I,J)|_{\prec i}}$. An interpretation I is a *stratified supported model* or *stratified stable model* of Π with respect to the splitting $(\Sigma_i)_{i\in I}$ iff for each $i \in I$, $I|_i$ is a supported model or stable model of $\Pi\langle J, J\rangle$, where J is $I|_{\cup_{j\prec i}\Sigma_j}$.

Let us illustrate these definition by computing the stratified well-founded model of our example program E.

Example 3.3. Recall that the program E is partitioned into the programs

$$\begin{aligned} E_0 =& \{\}, \\ E_1 =& \{P \leftarrow \neg Q, \neg R; Q \leftarrow \neg P, \neg R\}, \\ E_2 =& \{S \leftarrow P, Q\}. \end{aligned}$$

The well-founded model of E_0 is $(\{\}, \{\})$. Replacing the atom R in E_1 by its truth-value according to $\{\}$ yields the new program $E_1\langle\{\}, \{\}\rangle = \{P \leftarrow \neg Q \wedge \mathbf{t}; Q \leftarrow \neg P \wedge \mathbf{t}\}$. The well-founded model of this program is $(\{\}, \{P, Q\})$. For the component $(\mathcal{T}_E)_2^{(\{\},\{P,Q\})}$, we need to consider both the program $E_2' = E_2\langle\{\}, \{P, Q\}\rangle = \{\}$ and the program $E_2'' = E_2\langle\{P, Q\}, \{\}\rangle = \{S\}$. The component $(\mathcal{T}_E)_2^{(\{\},\{P,Q\})}$ is now $(U_{E_2'}, U_{E_2''})$. The well-founded fixpoint of this component is $(\{\}, \{S\})$. Therefore, the stratified well-founded model of E is:

$$(\{\} \cup \{\} \cup \{\}, \{\} \cup \{P, Q\} \cup \{S\}) = (\{\}, \{P, Q, S\}).$$

Let us now summarize the results of this section. The fundamental concept is that of a dependency relation for a logic program Π. Given such a dependency relation,

we can construct a splitting $(\Sigma_i)_{i\in I}$ of the alphabet of Π. We can then consider a stratified construction process, which proceeds along the well-founded order $\preceq$ on I and, at each level $i \in I$, constructs the model (under one of the semantics we consider) of the component-operator of $(\mathcal{T}_\Pi)_i^{(U,V)}$ on that level, using the programs $\Pi_i\langle U, V\rangle$ and $\Pi_i\langle V, U\rangle$ (or, equivalently, the program $\Pi_i\langle U, V\rangle_4$ containing four-valued truth values). The main result that we have proved in this section is that such a stratified construction process produces precisely the model of the program under the semantics in question. Formally, we can summarize this result as follows.

Theorem 3.5. *Let Π be a propositional logic program with a splitting $(\Sigma_i)_{i\in I}$. A pair of interpretations (I, J) is the Kripke-Kleene model (respectively, a partial stable model or the well-founded model) of Π iff (I, J) is the stratified Kripke-Kleene model (respectively, a stratified partial stable model or the stratified well-founded model) of Π with respect to $(\Sigma_i)_{i\in I}$. Moreover, an interpretation I is a supported model (respectively, exact stable model) of Π iff I is a stratified supported model (a stratified exact stable model) of Π with respect to $(\Sigma_i)_{i\in I}$.*

3.3.2 General rule sets

We now discuss how the previous results for propositional logic programs can be extended to the more general rule sets defined in Section 2.2. In the previous section, we started our analysis by showing that the lattice $\mathcal{I}_\Sigma$ of interpretations for the propositional alphabet Σ is isomorphic to the product lattice $\otimes_{p\in\Sigma}\mathcal{I}_{\{p\}}$. In the context of a rule set Δ, we get the following setting.

Let F be a pre-interpretation for the alphabet of Δ and let O_1, O_2 be interpretations of the open predicates of Δ that extend F, i.e., $O_1, O_2 \in \mathcal{L}^F_{Op(\Delta)}$. To study the operator $\mathcal{T}_\Delta^{(O_1,O_2)}$, we need to work in the lattice $\mathcal{L}^F_{Def(\Delta)}$, which is isomorphic to all possible ways of assigning either $\mathbf{t}$ or $\mathbf{f}$ to every domain atom $P(\mathbf{d}) \in \mathcal{A}t^F_{Def(\Delta)}$. So, $\mathcal{L}^F_{Def(\Delta)}$ is isomorphic to the product lattice $\otimes_{P(\mathbf{d})\in\mathcal{A}t^F_{\mathbf{P}}} L_{P(\mathbf{d})}$, where each $L_{P(\mathbf{d})}$ is simply $\{\mathbf{t}, \mathbf{f}\}$.

We can now define the concept of a dependency relation for a rule set.

Definition 3.12. Let Δ be a rule set, F a pre-interpretation for the alphabet of this rule set, and $O_1, O_2 \in \mathcal{L}^F_{Op(\Delta)}$. A binary relation $\rightsquigarrow$ on $\mathcal{A}t^F_{Def(\Delta)}$ is a dependency relation for Δ in (O_1, O_2) if for every domain atom $P(\mathbf{d})$ and all pairs $(I, J), (I', J') \in \mathcal{L}^F_{Def(\Delta)}$ such that $(I, J)|_{(\rightsquigarrow P(\mathbf{d}))} = (I', J')|_{(\rightsquigarrow P(\mathbf{d}))}$, the following condition holds: for every rule of the form $\forall\mathbf{x}\ P(\mathbf{t}) \leftarrow \varphi$, if $\mathbf{c}$ is a tuple of domain elements, such that $\mathbf{t}[\mathbf{x}/\mathbf{c}]^F = \mathbf{d}$, then

$$\varphi[\mathbf{x}/\mathbf{c}]^{(O_1\cup I, O_2\cup J)} = \varphi[\mathbf{x}/\mathbf{c}]^{(O_1\cup I', O_2\cup J')}.$$

Once again, it follows directly from the definition of $\mathcal{T}_\Delta^{(O_1,O_2)}$, that every such dependency relation $\rightsquigarrow$ of a rule set Δ in (O_1, O_2) is also a dependency relation of this operator. Let us now first discuss how such dependency relations can be constructed, before turning our attention to the question of how to use them.

Constructing dependency relations.

There is again on obvious way of construction a dependency relation for a rule set Δ. Namely, we can consider the relation $\rightsquigarrow^{ob}_{\Delta}$, that is defined as: $P(\mathbf{a}) \rightsquigarrow^{ob}_{\Delta} Q(\mathbf{c})$ iff there exists a rule $r \in \Delta$ such that the predicate symbol P appears in $body(r)$ and Q appears in $head(r)$. However, it is clear that this dependency relation can be quite a bit larger—and therefore less informative—than it actually needs to be. For instance, let us consider the following example.

Example 3.4. The even and odd natural numbers can be defined by the following rule set:

$$\Delta_{EvOd} = \left\{ \begin{array}{c} Even(0). \\ \forall n\ Even(n+1) \leftarrow Odd(n). \\ \forall n\ Odd(n+1) \leftarrow Even(n). \end{array} \right\}$$

We consider this definition in the natural pre-interpretation with domain $\mathbb{N}$. The obvious dependency relation $\rightsquigarrow^{ob}_{\Delta}$ now consists of all pairs $Even(n) \rightsquigarrow^{ob}_{\Delta} Odd(m)$ and $Odd(n) \rightsquigarrow^{ob}_{\Delta} Even(m)$, with $m, n \in \mathbb{N}$.

We will now present a way of constructing a smaller, more fine-grained dependency relation, which works on the level of individual domain atoms, instead of on the level of entire predicates. We first introduce the concept of a *base* for a formula φ. Intuitively, a base for φ is a set B of domain atoms, such that the truth value of all atoms in B completely determines the truth value of φ.

Definition 3.13. Let φ be a formula and F a pre-interpretation for its alphabet. A set of domain atoms B is a *base* for φ in F iff for all pairs $(I, J), (I', J')$ of interpretations extending F, if $I|_B = I'|_B$ and $J|_B = J'|_B$, then $\varphi^{(I,J)} = \varphi^{(I',J')}$.

Clearly, any superset of a base is also a base. The problem of finding a dependency relation for a definition Δ can be reduced to that of finding bases for bodies of rules, as shown by the following trivial proposition.

Proposition 3.10. *Let Δ be a definition and let $\rightsquigarrow$ be a binary relation on $\mathcal{A}t^F_{Def(\Delta)}$. Suppose that for all $I, J \in \mathcal{L}^F_{Def(\Delta)}$, for every rule $\forall \mathbf{x}\ P(\mathbf{t}) \leftarrow \varphi$ in Δ and every tuple $\mathbf{c}$, the set $(\rightsquigarrow P(\mathbf{t}[\mathbf{x}/\mathbf{c}]^F))$ is a base for $\varphi[\mathbf{x}/\mathbf{c}]$ in F. Then $\rightsquigarrow$ is a dependency relation of Δ in any pair $(O_1, O_2) \in (\mathcal{L}^F_{Op(\Delta)})^2$.*

We now show how we can construct such a base for a formula φ.

Definition 3.14. Let φ be a formula and F a pre-interpretation for its alphabet. We inductively define a base $B_F(\varphi)$ as follows:

- For all $P(\mathbf{t})$, $B_F(P(\mathbf{t})) = \{P(\mathbf{t}^F)\}$;
- for all $(\varphi_1 \vee \varphi_2)$, $B_F(\varphi_1 \vee \varphi_2) = B_F(\varphi_1) \cup B_F(\varphi_2)$;
- for all $(\exists x\ \varphi)$, $B_F(\exists x\ \varphi) = \bigcup_{d \in D} B_F(\varphi[x/d])$;
- for all $(\neg\varphi)$: $B_F(\neg\varphi) = B_F(\varphi)$.

It can easily be seen that, for each φ, $B_F(\varphi)$ is indeed a base of φ in F. Using this definition, we can now construct a more refined dependency relation for a rule set Δ.

Definition 3.15. Let Δ be a rule set and F a pre-interpretation for its alphabet. We define the binary relation $\rightsquigarrow^F_\Delta$ on $\mathcal{A}t^F_{Def(\Delta)}$ as follows: $P(\mathbf{a}) \rightsquigarrow^F_\Delta Q(\mathbf{c})$ iff Δ contains a rule $\forall \mathbf{x}\ Q(\mathbf{t}) \leftarrow \varphi$ for which there exists a tuple $\mathbf{e}$ such that $\mathbf{t}[\mathbf{x}/\mathbf{e}]^F = \mathbf{c}$ and $P(\mathbf{a}) \in B_F(\varphi[\mathbf{x}/\mathbf{e}])$.

For our example of even and odd numbers, this dependency relation now indeed gives us the wanted results, i.e., we can conclude that for all $n \in \mathbb{N}$, $Even(n) \rightsquigarrow^F_\Delta Odd(n+1)$ and $Odd(n) \rightsquigarrow^F_\Delta Even(n+1)$, and these are the only tuples that belong to this dependency relation. However, while this dependency relation $\rightsquigarrow^F_\Delta$ gives us the wanted result for this particular example, there are other cases in which it is still not as small as we might like. Let us illustrate this by the following example.

Example 3.5. Consider a game between two players who each take turns, removing either one or two stone(s) from a pile of n stones, such that the player who makes the last move wins. Given an appropriate interpretation for the open predicate $Move/2$, the winning positions of this game can be defined by the following rule:

$$\forall x\ Win(x) \leftarrow \exists y\ Move(x,y) \wedge \neg Win(y).$$

Even if we were to apply our more refined definition to this example, we would still not be able to discover anything, apart from that, for all moves m, n, $Win(m) \rightsquigarrow^F_\Delta Win(n)$. In order to be able to draw more useful conclusions in this case, we need to take into account the fact that $Move$ is an open predicate, whose interpretation is known beforehand. We first extend our notion of a base to also take into account open predicates. The basic idea is to consider a set $\mathcal{O}$ of predicates—this will be the open predicates of the definition—for which an interpretation (O_1, O_2) is already given.

Definition 3.16. Let φ be a formula in alphabet Σ, F a pre-interpretation for Σ and (O_1, O_2) a pair of interpretations for some set of predicates $\mathcal{O} \subseteq \Sigma^p$. A set of domain atoms B is a *$\mathcal{O}$-base* for φ in (O_1, O_2) iff for all pairs $(I, J), (I', J')$ of interpretations in $\mathcal{L}^F_{\Sigma^p \setminus \mathcal{O}}$, if $I|_B = I'|_B$ and $J|_B = J'|_B$, then $\varphi^{(O_1 \cup I, O_2 \cup J)} = \varphi^{(O_1 \cup I', O_2 \cup J')}$.

Let us now extend our method for constructing a base to this new setting. Since we now assume a fixed interpretation for the predicates in $\mathcal{O}$, no more dependencies will be generated by an atom $P(\mathbf{a})$ with $P \in \mathcal{O}$, or by any formula φ whose truth value is already completely determined by assigning this particular interpretation to the predicates $\mathcal{O}$. The following definition of a base $B^{\mathcal{O}}_{(O_1,O_2)}(\varphi)$ formalizes this. In a number of places, it distinguishes between formulas whose base is empty and formulas for which it is not. The idea is that the former kind of formulas are those whose truth is already fully determined by the interpretation assigned to $\mathcal{O}$. For such a formula φ with an empty base, we write $\varphi^{(O_1,O_2)}$ as a shorthand for the statement that for all $I, J \in \mathcal{L}^F_{\Sigma^p \setminus \mathcal{O}}$, $\varphi^{(O_1 \cup I, O_2 \cup J)} = \mathbf{t}$. Given the definition below, it will be an easy proof by induction to show that, whenever the base $B^{\mathcal{O}}_{(O_1,O_2)}(\varphi)$ is empty, this last statement is also equivalent to $\varphi^{(O_1 \cup I, O_2 \cup J)} = \mathbf{t}$ for *some* pair (I, J), i.e., it is indeed the case that I and J do not matter.

Definition 3.17. Let φ be a formula, $\mathcal{O}$ a set of predicates of the alphabet of this formula and (O_1, O_2) a pair of interpretations in $\mathcal{L}^F_{\mathcal{O}}$. We inductively define $B^{\mathcal{O}}_{(O_1,O_2)}(\varphi)$ as follows:

- For all $P(\mathbf{t})$, such that $P \in \mathcal{O}$, $B^{\mathcal{O}}_{(O_1,O_2)}(P(\mathbf{t})) = \{\}$;
- for all other $P(\mathbf{t})$, $B^{\mathcal{O}}_{(O_1,O_2)}(P(\mathbf{t})) = \{P(\mathbf{t}^F)\}$.
- For all $(\varphi_1 \vee \varphi_2)$, such that, for $i = 1$ or $i = 2$, $B^{\mathcal{O}}_{(O_1,O_2)}(\varphi_i) = \{\}$ and $\varphi_i^{(O_1,O_2)} = \mathbf{t}$: $B^{\mathcal{O}}_{(O_1,O_2)}(\varphi_1 \vee \varphi_2) = \{\}$;
- for all other $(\varphi_1 \vee \varphi_2)$: $B^{\mathcal{O}}_{(O_1,O_2)}(\varphi_1 \vee \varphi_2) = B^{\mathcal{O}}_{(O_1,O_2)}(\varphi_1) \cup B^{\mathcal{O}}_{(O_1,O_2)}(\varphi_2)$.
- For all $(\exists x\, \varphi)$, such that for some $c \in D$, $B^{\mathcal{O}}_{(O_1,O_2)}(\varphi[x/c]) = \{\}$ and $\varphi[x/c]^{(O_1,O_2)} = \mathbf{t}$: $B^{\mathcal{O}}_{(O_1,O_2)}(\exists x\, \varphi) = \{\}$;
- for all other $(\exists x\, \varphi)$: $B^{\mathcal{O}}_{(O_1,O_2)}(\exists x\, \varphi) = \bigcup_{d \in D} B^{\mathcal{O}}_{(O_1,O_2)}(\varphi[x/d])$.
- For all $(\neg\varphi)$: $B^{\mathcal{O}}_{(O_1,O_2)}(\neg\varphi) = B^{\mathcal{O}}_{(O_2,O_1)}(\varphi)$.

The following result is now obvious.

Proposition 3.11. *Let φ be a formula, $\mathcal{O}$ a set of predicates of its alphabet and $O_1, O_2 \in \mathcal{L}^F_{\mathcal{O}}$. Then $B^{\mathcal{O}}_{(O_1,O_2)}(\varphi)$ is an $\mathcal{O}$-base for φ in (O_1, O_2).*

We can now again use this result to construct a dependency relation for a rule set, using the bases of its rule bodies.

Definition 3.18. Let Δ be a definition, F a pre-interpretation, and $O_1, O_2 \in \mathcal{L}^F_{Op(\Delta)}$. We define the binary relation $\rightsquigarrow^{(O_1,O_2)}_{\Delta}$ on domain atoms as: $P(\mathbf{a}) \rightsquigarrow^{(O_1,O_2)}_{\Delta} Q(\mathbf{c})$ iff Δ contains a rule $\forall \mathbf{x}\, Q(\mathbf{t}) \leftarrow \varphi$ for which there exists a tuple $\mathbf{e}$ such that $\mathbf{t}[\mathbf{x}/\mathbf{e}]^F = \mathbf{c}$ and $P(\mathbf{a}) \in B^{Op(\Delta)}_{(O_1,O_2)}(\varphi[\mathbf{x}/\mathbf{e}])$.

It is easy to see that any such $\rightsquigarrow^{(O_1,O_2)}_{\Delta}$ is indeed a dependency relation of Δ in (O_1, O_2).

Let us now consider again our rule r defining the winning moves of the stones-game. We consider a pre-interpretation F with domain some subset $[0, n]$ of $\mathbb{N}$. Let O interpret the open predicate $Move/2$ as dictated by our description of the game, i.e., $Move^O$ contains all pairs (i, j) for which $0 \leq i, j \leq n$ and either $j = i-1$ or $j = i-2$. If we then consider the dependency relation $\rightsquigarrow^{(O,O)}_{\{r\}}$ for this rule, we find that, indeed, $Win(j) \rightsquigarrow^{(O,O)}_{\Delta} Win(i)$ iff $j = i - 1$ or $j = i - 2$.

Using dependency relations

In Section 3.3.1 on propositional logic programs, we showed that, given a dependency relation for such a program Π, we can construct a splitting $(\Sigma_i)_{i\in I}$ of its alphabet and a corresponding partition $(\Pi_i)_{i\in I}$ of Π, from which we can then construct the components of the operator $\mathcal{T}_\Pi$ by means of a syntactical transformation. The components of $\mathcal{T}_\Delta^{(O_1,O_2)}$ can be constructed in a similar way. Rather than go through all the details of this, we will only sketch how the previous construction process needs to be adapted. In the case of general rule sets, a dependency relation gives us a splitting $(At_i)_{i\in I}$ of the set of domain atoms $\mathcal{A}t^F_{Def(\Delta)}$. In order to construct corresponding rule sets $(\Delta_i)_{i\in I}$, we first need to ground the rule set Δ. This means that we might need to extend the alphabet of the rule set in order to make sure that all domain elements are referenced by some constant or ground term. After this has been done, we can construct a grounding of our theory, by replacing variables by ground terms in all possible ways. This grounding might be an infinite set of possibly infinitary formulas. However, it is easy to switch to an infinitary extension of our formalism, in which this does not pose a problem. From this grounding, we can then derive the component operators by replacing ground atoms by $\mathbf{t}$ or $\mathbf{f}$ in the same way as for propositional logic programs.

Obviously, this process can only be performed in practice if the grounding of the program is finite. However, the current generation of Answer Set Programming model generators, such as SModels (Niemelä, Simons, and Syrjänen 2000) and DLV (Dell'Armi, Faber, Ielpa, Koch, Leone, Perri, and Pfeifer 2001), as well as the ID-logic model expansion system MIDL (Mariën, Mitra, Denecker, and Bruynooghe 2005), all already require this property anyway. This means to the process described above is applicable to any program that could serve as an input to one of these systems. Moreover, since all of these programs completely compute the grounding before starting the model generation phase, it is possible to include an analysis of modularity properties in between these two phases at minimal cost. This might be useful not only to provide interesting feedback to the user about the structure of the program, but also because information about the modularity of a theory can be used to speed up model generation, as already explained in Section 3.1.

Even when it is not practically feasible to construct the grounding of a program, we might still be able draw some useful conclusions from our modularity results. The following section presents an example of this, in the context of ID-logic.

Splitting definitions in ID-logic

In ID-logic, we can use dependency relations to split a big definition into a conjunction of smaller definitions. Let us illustrate this by an example. We have already encountered the rule set Δ_{EvOd} that defines the even and odd natural numbers by mutual recursion. In ID-logic, one could, however, also try to define these concepts in a different way, using the following two definitions, one of which defines the concept $Even$ in terms of an open predicate Odd, while the other defines Odd in terms of the an

predicate $Even$:

$$\Delta_{Ev} = \left\{ \begin{array}{ll} Even(0) & . \\ \forall n\ Even(n+1) & \leftarrow Odd(n). \end{array} \right\}$$
$$\Delta_{Od} = \{\forall n\ Odd(n+1) \leftarrow Even(n).\}$$

Now, this raises the obvious question of whether the conjunction $\Delta_{Ev} \wedge \Delta_{Od}$ is equivalent to the simultaneous definition Δ_{EvOd} of these two concepts by mutual recursion.

We can answer this question by using our algebraic results on recombinations, which were presented in Section 3.2.5. Let us first recall that the semantics of ID-logic states that an interpretation I is a model of a definition Δ iff $I|_{Def(\Delta)}$ is the well-founded fixpoint of the operator $\mathcal{T}_\Delta^{(I,I)|_{Op(\Delta)}}$. To ease notation, let us denote such an operator of the form $\mathcal{T}_\Delta^{(O,O)}$ as simply $\mathcal{T}_\Delta^O$. We will now show that, for every structure I, the operators $\mathcal{T}_{\Delta_{Ev}}^{I|_{Odd}}$ and $\mathcal{T}_{\Delta_{Od}}^{I|_{Even}}$ of these two definitions are recombinations of the operator $\mathcal{T}_{\Delta_{EvOd}}$ of the original definition Δ. In general, we will partition a big definition Δ into a set of smaller definitions $\{\Delta_1, \ldots, \Delta_n\}$, in such a way that, first, all rules for the same predicate belong to the same Δ_j and, second, there is some splitting $(At_i)_{i\in I}$ for Δ, such that all rules for atoms from the same At_i also belong to the same Δ_j.

Definition 3.19. Let Δ be a definition and let $(At_i)_{i\in I}$ be a splitting for Δ. A partition $\{\Delta_1, \ldots, \Delta_n\}$ of Δ *respects* the splitting $(At_i)_{i\in I}$ iff the following two conditions hold:

- For all $j, j' \in 1..n$, if $j \neq j'$, then $Def(\Delta_j) \cap Def(\Delta_{j'}) = \{\}$;
- For all $i \in I$ and $P(\mathbf{a}), Q(\mathbf{c}) \in At_i$, there exists a $1 \leq j \leq n$, such that $P, Q \in Def(\Delta_j)$.

We can now relate such a partition of Δ to recombinations of the operator $\mathcal{T}_\Delta^O$ as follows.

Proposition 3.12. *Let Δ be a definition and $\{\Delta_1, \ldots, \Delta_n\}$ a partition of Δ that respects some splitting $(At_i)_{i\in I}$ for Δ. For some $1 \leq j \leq n$, let O interpret $Op(\Delta_j)$ and let J be the set of all $i \in I$ for which there exists a domain atom $P(\mathbf{a}) \in \Sigma_i$ such that $P \in Def(\Delta_j)$. Then $\mathcal{T}_{\Delta_j}^O$ is equal to the recombination $(\mathcal{T}_\Delta^{O_1})_J^{O_2}$, with $O_1 = O|_{Op(\Delta)}$ and $O_2 = O|_{Op(\Delta_j)\setminus Op(\Delta)}$.*

Proof. Let $\mathcal{T}_{\Delta_j}^O$ and $\mathcal{T}_\Delta^{O_1}$ be as above. Because the partition $\{\Delta_1, \ldots, \Delta_n\}$ respects a splitting for for Δ, the operator $\mathcal{T}_\Delta^{O_1}$ now indeed has a recombination $(\mathcal{T}_\Delta^{O_1})_J^{O_2}$ with J and O_2 as above. Moreover, the domain of this recombination is indeed $\mathcal{L}_{Def(\Delta_j)}^F$. It now follows directly from the definitions of the two operators, that $\mathcal{T}_{\Delta_j}^O = (\mathcal{T}_\Delta^{O_1})_J^{O_2}$ iff for all interpretation I, J of $Def(\Delta_j)$, the following two statements are equivalent:

- There exists a rule $\forall \mathbf{x}\ P(\mathbf{t}) \leftarrow \varphi$ in Δ_j, for which there exists a $\mathbf{c}$, such that $(O \cup I, O \cup J) \models \varphi[\mathbf{x}/\mathbf{c}]$.

- There exists a rule $\forall \mathbf{x}\ P(\mathbf{t}) \leftarrow \varphi$ in Δ, for which there exists a $\mathbf{c}$, such that $(O_1 \cup O_2 \cup I, O_1 \cup O_2 \cup J) \models \varphi[\mathbf{x}/\mathbf{c}]$.

Because, for each $P \in Def(\Delta_j)$, Δ_j contains precisely all rules from Δ with predicate P in their head, this is the case.

□

As a direct consequence of this proposition and Theorem 3.3, we now have the following equivalence result.

Theorem 3.6. *Let Δ be a definition, O an interpretation of $Op(\Delta)$ and $\{\Delta_1, \ldots, \Delta_n\}$ a partition of Δ that respects some splitting for Δ. Then for each structure S extending O:*

$$S \models \Delta \textit{ iff } S \models \Delta_1 \wedge \cdots \wedge \Delta_n.$$

So, for our example, it is indeed the case that the conjunction $\Delta_{Ev} \wedge \Delta_{Od}$ of the two separate definitions of $Even$ and Odd is equivalent to the definition Δ_{EvOd} by mutual recursion.

3.3.3 Related work

(Lifschitz and Turner 1994) proved a splitting theorem for propositional logic programs under the stable model semantics; similar results were independently obtained by (Eiter, Gottlob, and Mannila 1997). These results were proven for programs with a syntax that is not completely subsumed by ours. Concretely, they allow disjunction in the head of clauses and the use of two kinds of negation (negation-as-failure and classical negation). While our results could easily be extended to incorporate the two negations, the extension to disjunction in the head is less straightforward. However, the fact that the stable model semantics for disjunctive logic programs can also be characterized as a fixpoint semantics (Leone, Rullo, and Scarcello 1995), seems to suggest that our approach could be used to obtain similar results for this extended syntax as well. Indeed, there has already been some work into extending approximation theory to also capture the semantics of this kind of programs (Pelov and Truszczyński 2004), which could be useful for this.

Even though our syntax is more restricted than that of Lifschitz et al. and of Eiter et al., our results are, in some important respects, more general than theirs. Indeed, first, our results apply not only to the stable semantics, but also to supported model, Kripke-Kleene and well-founded semantics. Second, the rule sets we consider allow arbitrary formulas in rule bodies, do not fix the domain to the Herbrand universe, and allow open predicates. These features make our results also applicable to extensions of logic programming, that are not covered by their results, such as abductive logic programs or ID-logic. Among the results that we have proven for ID-logic, there is a theorem that allows a definition to be split into any partition that respects a dependency relation for this definition. In (Denecker and Ternovska 2004), we find a theorem that corresponds to the restriction of this result to those cases where each Δ_j is total given O. Our theorem is strictly more general. Earlier work by (Verbaeten, Denecker, and Schreye 2000) studies modularity of normal logic programs with open predicates

under the well-founded semantics, by means of the theory of *justifications*. The syntactical criteria they derive from their semantical analysis coincide with those we have presented in this section.

In order to further motivate and explain the well-founded model semantics, (Przymusinski 1998) defined the *dynamic stratification* of a program. The level of an atom in this stratification is based on the number of iterations it takes the Gelfond-Lifschitz operator to determine the truth-value of this atom.[1] As such, this stratification precisely mimics the computation of the well-founded model and is, therefore, the tightest possible stratification of a program under the well-founded semantics. However, as there exist no syntactic criteria which can be used to determine whether a certain stratification is the dynamic stratification of a program—in fact, the only way of deciding this is by actually constructing the well-founded model of the program—this concept cannot be used to perform the kind of static, upfront splitting which is our goal.

3.4 Application to autoepistemic logic

Applying our algebraic results to autoepistemic logic is somewhat less straightforward than it was for logic programs. Let us first explain why this is the case. Let $(\Sigma_i)_{i\in I}$ be a partition of the alphabet Σ, with $\langle I, \preceq\rangle$ a well-founded index set. For an interpretation $X \in \mathcal{I}_\Sigma$, we denote the intersection $X \cap \Sigma_i$ by $X|_{\Sigma_i}$. For a possible world structure Q, $\{X|_{\Sigma_i} \mid X \in Q\}$ is denoted by $Q|_{\Sigma_i}$.

In Section 2.3, we defined the semantics of autoepistemic logic in terms of an operator on the bilattice $\mathcal{B}_\Sigma = \mathcal{W}_\Sigma^2$. However, for our purpose of stratifying autoepistemic theories, we are interested in the bilattice $\tilde{\mathcal{B}}_\Sigma$ of the product lattice $\tilde{\mathcal{W}}_\Sigma = \otimes_{i\in I}\mathcal{W}_{\Sigma_i}$. An element of this product lattice consists of a number of possible interpretations for each level Σ_i. As such, if we choose for each Σ_i one of its interpretations, the union of these chosen interpretations is an interpretation for the entire alphabet Σ. Therefore, the set of all possible ways of choosing one interpretation for each Σ_i determines a set of possible interpretations for Σ, i.e., an element of $\mathcal{W}_\Sigma$. More formally, we define:

$$\kappa : \tilde{\mathcal{W}}_\Sigma \to \mathcal{W}_\Sigma : \tilde{Q} \mapsto \{\bigcup_{i\in I} S(i) \mid S \in \otimes_{i\in I}\tilde{Q}(i)\}.$$

Similarly, $\tilde{\mathcal{B}}_\Sigma$ can be mapped to $\mathcal{B}_\Sigma$ by the function $\overline{\kappa}$, which maps each $(\tilde{P}, \tilde{S}) \in \tilde{\mathcal{B}}_\Sigma$ to $(\kappa(\tilde{P}), \kappa(\tilde{S}))$.

This function κ is, however, not an isomorphism. Indeed, unlike $\mathcal{W}_\Sigma$, elements of $\tilde{\mathcal{W}}_\Sigma$ cannot express that an interpretation for a level Σ_i is possible in combination with a *certain* interpretation for another level Σ_j, but *not* with a different interpretation for Σ_j. For instance, if we split an alphabet $\Sigma = \{p, q\}$ into $\Sigma_0 = \{p\}$ and $\Sigma_1 = \{q\}$, the element $\{\{p, q\}, \{\}\}$ of $\mathcal{W}_\Sigma$ is not in $\kappa(\tilde{\mathcal{W}}_\Sigma)$, because it expresses that $\{p\}$ is only a possible interpretation for Σ_0 when Σ_1 is interpreted by $\{q\}$ and not when Σ_1 is interpreted by $\{\}$. To make this more precise, we introduce the following concept of a possible world structure Q being *disconnected* w.r.t. a certain partition $(\Sigma_i)_{i\in I}$ of

[1] To be a bit more precise, p belongs to level Σ_i iff i is the minimal j for which $\mathcal{GL}^j(\bot, \top) = (I, J)$ and either $p \in I$ or $p \notin J$.

the alphabet. Intuitively, this is the case if, whenever an interpretation X_i for Σ_i is possible in combination with some interpretation Y_j for Σ_j, then X_i is also possible in combination with every other interpretation Y'_j for Σ_j appearing in Q.

Definition 3.20. A possible world structure $Q \in \mathcal{W}_\Sigma$ is *disconnected* w.r.t. a partition $(\Sigma_i)_{i\in I}$ of its alphabet iff for all possible worlds $X, Y \in Q$ and for each $i \in I$, $(Y|_{\Sigma_i} \cup \bigcup_{j\neq i} X|_{\Sigma_j}) \in Q$.

It is now obvious that we can now characterize the image of κ of follows.

Proposition 3.13. *Let $(\Sigma_i)_{i\in I}$ be a partition of an alphabet Σ and let $\tilde{\mathcal{W}}_\Sigma$ be $\otimes_{i\in I}\mathcal{W}_{\Sigma_i}$. The image of $\kappa : \tilde{\mathcal{W}}_\Sigma \to \mathcal{W}_\Sigma$ can be characterized as follows:*

$$\kappa(\tilde{\mathcal{W}}_\Sigma) = \{Q \in \mathcal{W}_\Sigma \mid Q \text{ is disconnected w.r.t. } (\Sigma_i)_{i\in I}\}.$$

In order to achieve our goal of being able to incrementally construct the models of a theory by means of the components of some operator on $\tilde{\mathcal{B}}_\Sigma$, we now need to restrict our attention to a class of theories whose models are all disconnected. We will define this class using a concept of a dependency relation for autoepistemic logic.

Definition 3.21. A dependency relation for an autoepistemic theory T in alphabet Σ is a binary relation $\rightsquigarrow$ on Σ that satisfies the following condition: for every formula $\varphi \in T$ and all atoms p, q such that $p \in At_O(\varphi)$ and $q \in At(\varphi)$, $q \rightsquigarrow p$.

To illustrate, let us consider the following example:

$$F = \{p \vee \neg Kp; K(p \vee q) \vee q\}.$$

Clearly, the binary relation on the alphabet $\{p, q\}$, consisting of tuples $p \rightsquigarrow p$, $p \rightsquigarrow q$ and $q \rightsquigarrow q$, is a dependency relation for F.

We can now again use such a dependency relation to stratify the alphabet of a theory.

Definition 3.22. Let T be an autoepistemic theory and $\langle I, \preceq\rangle$ a well-founded poset. T is *stratifiable* with respect to a partition $(\Sigma_i)_{i\in I}$ of its alphabet iff there exists a dependency relation $\rightsquigarrow$ for T, such that $(\Sigma_i)_{i\in I}$ respects $\rightsquigarrow$, i.e., whenever $p \leq_\rightsquigarrow q$ for some $p \in \Sigma_i$ and $q \in \Sigma_j$, then $i \preceq j$.

It is easy to see that such a stratifiable theory T can be split into a corresponding partition $(T_i)_{i\in I}$, such that for each $i \in I$ and $\varphi \in T_i$: $At_O(\varphi) \subseteq \Sigma_i$ and $At(\varphi) \subseteq \bigcup_{j\prec i} \Sigma_j$. For instance, our example theory F is stratifiable w.r.t. the partition $\Sigma_0 = \{p\}$, $\Sigma_1 = \{q\}$ of its alphabet $\{p, q\}$ and the corresponding partition of F is $F_0 = \{p \vee \neg Kp\}$, $F_1 = \{K(p \vee q) \vee q\}$.

Clearly, for a stratifiable theory, the evaluation $\mathcal{H}_{(P,S),X}(\varphi)$ of a formula $\varphi \in T_i$ only depends on the value of (P, S) in strata $j \preceq i$ and that of X in stratum i.

Proposition 3.14. *Let T be a stratifiable autoepistemic theory. Let $i \in I$ and $\varphi \in T_i$. Then for each (P, S), $(P', S') \in \mathcal{B}_\Sigma$ and $X, X' \in \mathcal{I}_\Sigma$, such that $X|_{\Sigma_i} = X'|_{\Sigma_i}$ and $P|_{\cup_{j\prec i}\Sigma_j} = P'|_{\cup_{j\prec i}\Sigma_j}$ and $S|_{\cup_{j\prec i}\Sigma_j} = S'|_{\cup_{j\prec i}\Sigma_j}$, $\mathcal{H}_{(P,S),X}(\varphi) = \mathcal{H}_{(P',S'),X'}(\varphi)$.*

This proposition can now be used to show that only disconnected belief pairs are relevant for the operator $\mathcal{D}_T$.

Proposition 3.15. *Let T be a stratifiable autoepistemic theory. Then each $\mathcal{D}_T(P,S)$ is disconnected.*

Proof. Let $(P,S) \in \mathcal{B}_\Sigma$ and $X, Y \in \mathcal{D}^u_T(P,S)$. By proposition 3.14, for each $Z \in \mathcal{I}_\Sigma$, such that, for some $i \in I$, $Z|_{\Sigma_i} = X|_{\Sigma_i}$ and, $\forall j \neq i$, $Z|_{\Sigma_j} = Y|_{\Sigma_j}$, $Z \in \mathcal{D}^u_T(P,S)$. Therefore $\mathcal{D}^u_T(P,S)$ and $\mathcal{D}_T(P,S)$ are both disconnected. □

This result suggests that the fact that κ is not surjective should not pose any problems, because we can simply forget about possible world structures of $\mathcal{W}_\Sigma$ that do not correspond to elements of our product lattice $\tilde{\mathcal{W}}_\Sigma$. However, there is another difference between the lattices $\tilde{\mathcal{W}}_\Sigma$ and $\mathcal{W}_\Sigma$, that also needs to be taken into account. Indeed, besides not being surjective, κ is also not injective. Concretely, $\tilde{\mathcal{W}}_\Sigma$ contains multiple "copies" of the empty set, that is, for any $\tilde{Q} \in \tilde{\mathcal{W}}_\Sigma$, as soon as for some $i \in I$, $\tilde{Q}(i) = \{\}$, it is the case that $\kappa(\tilde{Q}) = \{\}$.

Let us introduce some notation and terminology. We call a possible world structure $Q \in \mathcal{W}_\Sigma$ *consistent* if $Q \neq \{\}$; the set of all consistent Q is denoted by $\mathcal{W}^c_\Sigma$. A belief pair (P,S) is called consistent if both P and S are consistent; the set of all consistent belief pairs is denoted $\mathcal{B}^c_\Sigma$. Similarly, a $\tilde{Q} \in \tilde{\mathcal{W}}_\Sigma$ is called consistent if $\kappa(\tilde{Q}) \neq \{\}$ and the set of all consistent $\tilde{Q}$ is denoted as $\tilde{\mathcal{W}}^c_\Sigma$. Finally, a belief pair $(\tilde{P}, \tilde{S}) \in \tilde{\mathcal{B}}_\Sigma$ is called consistent if both $\kappa(\tilde{P}) \neq \{\}$ and $\kappa(\tilde{S}) \neq \{\}$ and we denote the set of all consistent pairs $(\tilde{P}, \tilde{S})$ as $\tilde{\mathcal{B}}^c_\Sigma$.

We will often need to eliminate inconsistent possible world structures from our considerations. Intuitively, the reason for this is that, when constructing a stratification, we need every stratum i to be completely independent of all strata j for which $j \not\preceq i$. However, if an inconsistency occurs at level j, then this could affect the way in which a lower level i is interpreted, because it will eliminate *all* possible worlds. Mathematically, this problem manifest itself by the fact that the equality $\kappa(\tilde{Q})|_{\preceq i} = \kappa(\tilde{Q}|_{\preceq i})$ only holds for consistent possible world structures $\tilde{Q}$.

We now summarize some obvious properties of κ.

Proposition 3.16. *The function κ has the following properties.*

1. *κ is order preserving;*
2. *κ is an embedding of $\tilde{\mathcal{W}}^c_\Sigma$ into $\mathcal{W}^c_\Sigma$ and an isomorphism between $\tilde{\mathcal{W}}^c_\Sigma$ and the set of all disconnected possible world structures in $\mathcal{W}^c_\Sigma$;*
3. *For all consistent $\tilde{Q}$, $\kappa(\tilde{Q}|_{\preceq i}) = \kappa(\tilde{Q})|_{\preceq i}$.*

Because of the differences between the lattices $\tilde{\mathcal{B}}_\Sigma$ and $\mathcal{B}_\Sigma$ outlined above, we cannot directly stratify the operator $\mathcal{D}_T$. Instead, we will define an intermediate operator $\tilde{\mathcal{D}}_T$ on $\tilde{\mathcal{B}}_\Sigma$, which is stratifiable by construction and whose consistent fixpoints are related to consistent fixpoints of $\mathcal{D}_T$. We define this operator $\tilde{\mathcal{D}}_T$ in such a way that, for any belief pair $(\tilde{P}, \tilde{S}) \in \tilde{\mathcal{B}}_\Sigma$, the ith level of $\tilde{\mathcal{D}}_T(\tilde{P}, \tilde{S})$ will be constructed using only the theory T_i and the restriction $(\tilde{P}, \tilde{S})|_{\preceq i}$ of $(\tilde{P}, \tilde{S})$.

Definition 3.23. Let T be a stratifiable autoepistemic theory. Let $(\tilde{P}, \tilde{S})$ be in $\tilde{\mathcal{B}}_\Sigma$. We define $\tilde{\mathcal{D}}^u_T(\tilde{P}, \tilde{S}) = \tilde{Q}$, with for each $i \in I$:

$$\tilde{Q}(i) = \{X \in \mathcal{I}_{\Sigma_i} \mid \forall \varphi \in T_i : \mathcal{H}_{\overline{\kappa}(\tilde{P}|_{\preceq i}, \tilde{S}|_{\preceq i}),X}(\varphi) = \mathbf{t}\}.$$

Furthermore, $\tilde{\mathcal{D}}_T(\tilde{P}, \tilde{S}) = (\tilde{\mathcal{D}}^u_T(\tilde{S}, \tilde{P}), \tilde{\mathcal{D}}^u_T(\tilde{P}, \tilde{S}))$ and $\tilde{D}_T(\tilde{Q}) = \tilde{\mathcal{D}}^u_T(\tilde{Q}, \tilde{Q})$.

Observe that we could equivalently define the ith level of $\tilde{\mathcal{D}}^u_T(\tilde{P}, \tilde{S})$ as the set $Mod(T_i\langle\overline{\kappa}(\tilde{P}|_{\preceq i}, \tilde{S}|_{\preceq i})\rangle)$ of models of the propositional theory $T_i\langle\overline{\kappa}(\tilde{P}|_{\preceq i}, \tilde{S}|_{\preceq i})\rangle$.

Let us now first show that, like its counterpart $\mathcal{D}_T$, this operator is also an approximation.

Proposition 3.17. *Let T be a stratifiable autoepistemic theory. Then $\tilde{\mathcal{D}}_T$ is an approximation.*

Proof. Let $(\tilde{P}, \tilde{S}), (\tilde{P}', \tilde{S}') \in \tilde{\mathcal{B}}_\Sigma$, such that $(\tilde{P}, \tilde{S}) \leq_p (\tilde{P}', \tilde{S}')$. By symmetry of $\tilde{\mathcal{D}}_T$, it suffices to show that $\tilde{\mathcal{D}}^u_T(\tilde{P}, \tilde{S}) \geq_k \tilde{\mathcal{D}}^u_T(\tilde{P}', \tilde{S}')$. Let $i \in I$. Because κ is order-preserving, $\overline{\kappa}(\tilde{P}|_{\preceq i}, \tilde{S}|_{\preceq i}) \leq_p \overline{\kappa}(\tilde{P}'|_{\preceq i}, \tilde{S}'|_{\preceq i})$. From (Denecker, Marek, and Truszczyński 2003), we know this implies that for each φ of T_i and X in $\mathcal{I}_{\Sigma_i}$, we have $\mathcal{H}_{\overline{\kappa}(\tilde{P}|_{\preceq i}, \tilde{S}|_{\preceq i}),X}(\varphi) \leq_t \mathcal{H}_{\overline{\kappa}(\tilde{P}'|_{\preceq i}, \tilde{S}'|_{\preceq i}),X}(\varphi)$. Hence, $\tilde{\mathcal{D}}^u_T(\tilde{P}, \tilde{S})(i) \subseteq \tilde{\mathcal{D}}^u_T(\tilde{P}', \tilde{S}')(i)$. □

We can relate the consistent fixpoints of $\tilde{\mathcal{D}}_T$ to those of $\mathcal{D}_T$, using the following result.

Proposition 3.18. *For all consistent $(\tilde{P}, \tilde{S}) \in \tilde{\mathcal{B}}^c_\Sigma$, $\overline{\kappa}(\tilde{\mathcal{D}}_T(\tilde{P}, \tilde{S})) = \mathcal{D}_T(\overline{\kappa}(\tilde{P}, \tilde{S}))$.*

Proof. Let $(\tilde{P}, \tilde{S}) \in \tilde{\mathcal{B}}^c_\Sigma$. By symmetry of the operators $\tilde{\mathcal{D}}_T$ and $\mathcal{D}_T$, it suffices to show that $\kappa(\tilde{\mathcal{D}}^u_T(\tilde{P}, \tilde{S})) = \mathcal{D}^u_T(\overline{\kappa}(\tilde{P}, \tilde{S}))$. Because for $i \neq j$, the objective atoms of T_i and T_j are disjoint, it is a trivial property of propositional logic that $Mod(T\langle\overline{\kappa}(\tilde{P}, \tilde{S})\rangle$ consists precisely of all worlds of the form $\cup_i X_i$ for which $X_i \in Mod(T_i\langle\kappa(\tilde{P}), \kappa(\tilde{S})\rangle)$. Now, let $\tilde{Q}$ be the element of $\tilde{\mathcal{B}}_\Sigma$ that maps every $i \in I$ to $Mod(T_i\langle\kappa(\tilde{P}), \kappa(\tilde{S})\rangle)$. We then have that $\kappa(\tilde{Q}) = \mathcal{D}^u_T(\kappa(\tilde{P}), \kappa(\tilde{S}))$. It therefore suffices to show that $\tilde{Q} = \tilde{\mathcal{D}}^u_T(\tilde{P}, \tilde{S})$, that is, that for all $i \in I$, $T_i\langle\kappa(\tilde{P}), \kappa(\tilde{S})\rangle = T_i\langle\kappa(\tilde{P}|_{\preceq i}), \kappa(\tilde{S}|_{\preceq i})\rangle$. Because T_i contains only modal literals in the alphabet $\cup_{j \preceq i}\Sigma_j$, we already have that $T_i\langle\kappa(\tilde{P}), \kappa(\tilde{S})\rangle = T_i\langle\kappa(\tilde{P})|_{\preceq i}, \kappa(\tilde{S})|_{\preceq i}\rangle$. Because $(\tilde{P}, \tilde{S})$ is consistent, we also have that $(\kappa(\tilde{P})|_{\preceq i}, \kappa(\tilde{S})|_{\preceq i}) = (\kappa(\tilde{P}|_{\preceq i}), \kappa(\tilde{S}|_{\preceq i}))$, which proves the result. □

We already know that $\overline{\kappa}$ is an isomorphism between $\tilde{\mathcal{B}}^c_\Sigma$ and its image $\overline{\kappa}(\tilde{\mathcal{B}}^c_\Sigma)$, and that all consistent fixpoints of $\mathcal{D}_T$ belong to $\overline{\kappa}(\tilde{\mathcal{B}}^c_\Sigma)$. Therefore, the above Proposition 3.18 now directly implies the following result.

Proposition 3.19. *The set of all $\kappa(\tilde{P}, \tilde{S})$ for which $(\tilde{P}, \tilde{S})$ is a consistent fixpoint of $\tilde{\mathcal{D}}_T$ is equal to the set of all consistent fixpoints of $\mathcal{D}_T$.*

A similar correspondence also holds for the consistent stable fixpoints of these two operators. Our proof of this depends on the following result.

Proposition 3.20. *For all consistent $\tilde{S} \in \tilde{\mathcal{W}}^c_\Sigma$, $\kappa(C^\downarrow_{\tilde{\mathcal{D}}_T}(\tilde{S})) = C^\downarrow_{\mathcal{D}_T}(\kappa(\tilde{S}))$.*

Proof. Recall that the operator $C^{\downarrow}_{\mathcal{D}_T}$ is defined as mapping each S to *lfp*$(\mathcal{D}^u_T(\cdot, S))$ and, similarly, $C^{\downarrow}_{\tilde{\mathcal{D}}_T}$ maps each $\tilde{S}$ to *lfp*$(\tilde{\mathcal{D}}^u_T(\cdot, \tilde{S}))$. Let $\tilde{S} \in \tilde{\mathcal{W}}^c_\Sigma$ and $S = \kappa(\tilde{S})$. The values $C^{\downarrow}_{\mathcal{D}_T}(S)$ and $C^{\downarrow}_{\tilde{\mathcal{D}}_T}(\tilde{S})$ can be constructed as the limit of, respectively, the ascending sequences $(Q_n)_{0\leq n}$ and $(\tilde{Q}_n)_{0\leq n}$, defined as follows: Q_0 is the bottom element $\mathcal{I}_\Sigma$ of $\mathcal{W}_\Sigma$ and for every $n > 0$, $Q_n = \mathcal{D}^u_T(Q_{n-1}, S)$; similarly, $\tilde{Q}_0$ is the bottom element of $\tilde{\mathcal{W}}_\Sigma$, that is, for all $i \in I$, $\tilde{Q}_0(i) = \mathcal{I}_{\Sigma_i}$, and for all $n > 0$, $\tilde{Q}_n = \tilde{\mathcal{D}}^u_T(\tilde{Q}_{n-1}, \tilde{S})$. It suffices to show that, for all $n \in \mathbb{N}$, $Q_n = \kappa(\tilde{Q}_n)$.

We prove this by induction over n. For the base case, it is clear that $\kappa(\tilde{Q}_0) = Q_0$. Now, suppose that the equality holds for n. We have to show that applying κ to $\tilde{Q}_{n+1} = \tilde{\mathcal{D}}^u_T(\tilde{Q}_n, \tilde{S})$ will yield $Q_{n+1} = \mathcal{D}^u_T(Q_n, S)$. By the induction hypothesis, this last expression is equal to $\mathcal{D}^u_T(\kappa(\tilde{Q}_n), \kappa(\tilde{S}))$. We distinguish two cases. First, let us assume that $\tilde{Q}_n$ is consistent. By Proposition 3.18, it is then the case that $\kappa(\tilde{\mathcal{D}}^u_T(\tilde{Q}_n, \tilde{S})) = \mathcal{D}^u_T(\overline{\kappa}(\tilde{Q}_n, \tilde{S}))$, which is precisely what needs to be proven. Second, assume that $\kappa(\tilde{Q}_n) = Q_n = \{\}$. Because $\tilde{Q}_{n+1} \geq_k \tilde{Q}_n$ and κ is order-preserving, $\kappa(\tilde{Q}_{n+1}) = \{\}$. Moreover, because also $Q_{n+1} \geq_k Q_n = \{\}$, we have that $Q_{n+1} = \{\}$, which means that, here too, we get the desired equality $\kappa(\tilde{Q}_{n+1}) = Q_{n+1}$. □

Together with the fact that $\overline{\kappa}$ is an isomorphism between $\tilde{\mathcal{B}}^c_\Sigma$ and its image $\kappa(\tilde{\mathcal{B}}^c_\Sigma)$, which contains all consistent elements in the image of $\mathcal{D}_T$, this result now implies the following correspondence between consistent stable fixpoints.

Proposition 3.21. *The set of all $\kappa(\tilde{P}, \tilde{S})$ for which $(\tilde{P}, \tilde{S})$ is a consistent stable fixpoint of $\tilde{\mathcal{D}}_T$ is equal to the set of all consistent stable fixpoints of $\mathcal{D}_T$.*

We can now summarize the content of Propositions 3.19 and 3.21 as follows.

Theorem 3.7. *Let T be a stratifiable theory. The set of all $\overline{\kappa}(\tilde{P}, \tilde{S})$ for which $(\tilde{P}, \tilde{S})$ is a consistent fixpoint (consistent stable fixpoint, respectively) of $\tilde{\mathcal{D}}_T$ is equal to the set of all (P, S), for which (P, S) is a consistent fixpoint (consistent stable fixpoint) of $\mathcal{D}_T$.*

We now define a class of theories, for which (partial) expansions and (partial) extensions cannot be inconsistent.

Definition 3.24. An autoepistemic theory T is *permaconsistent* if every propositional theory T' that can be constructed from T by replacing all non-nested occurrences of modal literals by **t** or **f** is consistent.

Observe that, contrary to what the above definition might suggest, we do not actually need to check *every* assignment in order to determine whether a theory is permaconsistent. Indeed, it suffices to only consider the worst case assignment, in which every positive occurrence of a modal literal is replaced by **f** and every negative occurrence is replaced by **t**. Moreover, we can also check permaconsistency for every stratum separately, because, for a stratifiable theory T, T is permaconsistent iff for every $i \in I$, T_i is permaconsistent.

Clearly, for a permaconsistent theory T, every belief pair in the image of $\mathcal{D}_T$ or $\tilde{\mathcal{D}}_T$ is consistent and, therefore, all fixpoints or stable fixpoints of these operators must be consistent as well. As such, Theorem 3.7 implies that the fixpoints and stable fixpoints

of $\mathcal{D}_T$ and $\tilde{\mathcal{D}}_T$ coincide, which of course implies that also the least fixpoint and well-founded fixpoint of these two operators coincide. In summary, we obtain the following result.

Theorem 3.8. *Let T be a stratifiable and permaconsistent theory. The set of all $\overline{\kappa}(\tilde{P}, \tilde{S})$ for which $(\tilde{P}, \tilde{S})$ is a fixpoint, the Kripke-Kleene fixpoint, a stable fixpoint, or the well-founded fixpoint of $\tilde{\mathcal{D}}_T$ is equal to the set consisting of, respectively, all fixpoints, the Kripke-Kleene fixpoint, all stable fixpoints, or the well-founded fixpoint of $\mathcal{D}_T$.*

This property does not hold for theories that are not permaconsistent. We illustrate this by the following example.

Example 3.6. Let T be the theory $\{Kp \Leftarrow p; \neg K(p \vee q)\}$. This T is stratifiable with respect to the partition $\Sigma_0 = \{p\}$, $\Sigma_1 = \{q\}$, with the corresponding partition of T being $T_0 = \{Kp \Leftarrow p\}$ and $T_1 = \{\neg K(p \vee q)\}$. However, T is not permaconsistent, because, for instance, $\{\mathbf{t} \Leftarrow p; \neg\mathbf{t}\}$ is not consistent. The operator $\mathcal{D}_T$ now has as its Kripke-Kleene fixpoint the pair $(\{\{p, q\}\}, \{\})$, as can be seen from the following computation:

$$\begin{aligned}
(\bot_k, \top_k) &= (\quad \mathcal{I}_{\{p,q\}} \quad , \quad \{\} \quad) \\
\mathcal{D}_T(\bot_k, \top_k) &= (Mod(T\langle\{\}, \mathcal{I}_{\{p,q\}}\rangle), Mod(T\langle\mathcal{I}_{\{p,q\}}, \{\}\rangle)) \\
&= (\quad Mod(\mathbf{t} \Leftarrow p; \neg\mathbf{f}) \quad , \quad Mod(\mathbf{f} \Leftarrow p; \neg\mathbf{t}) \quad) \\
&= (\quad \mathcal{I}_{\{p,q\}} \quad , \quad \{\} \quad)
\end{aligned}$$

The Kripke-Kleene fixpoint of $\tilde{\mathcal{D}}_T$ can be constructed as follows:

$$\begin{aligned}
(\tilde{\bot}, \tilde{\top})(0) &= (\quad \mathcal{I}_{\{p\}} \quad , \quad \{\} \quad) \\
(1) &= (\quad \mathcal{I}_{\{q\}} \quad , \quad \{\} \quad) \\
\tilde{\mathcal{D}}_T(\tilde{\bot}, \tilde{\top})(0) &= (\quad Mod(T_0\langle\{\}, \mathcal{I}_{\{p\}}\rangle) \quad , \quad Mod(T_0\langle\mathcal{I}_{\{p\}}, \{\}\rangle) \quad) \\
&= (\quad Mod(\mathbf{t} \Leftarrow p) \quad , \quad Mod(\mathbf{f} \Leftarrow p) \quad) \\
&= (\quad \mathcal{I}_{\{p\}} \quad , \quad \{\{\}\} \quad) \\
(1) &= (\quad Mod(T_1\langle\{\}, \mathcal{I}_{\{p,q\}}\rangle) \quad , \quad Mod(T_1\langle\mathcal{I}_{\{p,q\}}, \{\}\rangle) \quad) \\
&= (\quad Mod(\neg\mathbf{f}) \quad , \quad Mod(\neg\mathbf{t}) \quad) \\
&= (\quad \mathcal{I}_q \quad , \quad \{\} \quad) \\
\tilde{\mathcal{D}}_T^2(\tilde{\bot}, \tilde{\top})(0) &= (\quad Mod(T_0\langle\{\{\}\}, \mathcal{I}_{\{p\}}\rangle) \quad , \quad Mod(T_0\langle\mathcal{I}_{\{p\}}, \{\{\}\}\rangle) \quad) \\
&= (\quad Mod(\mathbf{f} \Leftarrow p) \quad , \quad Mod(\mathbf{f} \Leftarrow p) \quad) \\
&= (\quad \{\{\}\} \quad , \quad \{\{\}\} \quad) \\
(1) &= (\quad Mod(T_1\langle\{\}, \mathcal{I}_{p,q}\rangle) \quad , \quad Mod(T_1\langle\mathcal{I}_{\{p,q\}}, \{\}\rangle) \quad) \\
&= (\quad \mathcal{I}_{\{q\}} \quad , \quad \{\} \quad) \\
\tilde{\mathcal{D}}_T^3(\tilde{\bot}, \tilde{\top})(0) &= (\quad Mod(T_0\langle\{\{\}\}, \{\{\}\}\rangle) \quad , \quad Mod(T_0\langle\{\{\}\}, \{\{\}\}\rangle) \quad) \\
&= (\quad \{\{\}\} \quad , \quad \{\{\}\} \quad) \\
(1) &= (Mod(T_1\langle\{\}, \{\{\}, \{q\}\}\rangle), Mod(T_1\langle\{\{\}, \{q\}\}, \{\}\rangle)) \\
&= (\quad \mathcal{I}_{\{q\}} \quad , \quad \{\} \quad)
\end{aligned}$$

So, we find that applying κ to the Kripke-Kleene fixpoint of $\tilde{\mathcal{D}}_T$ yields $(\{\{\}, \{q\}\}, \{\})$, which is not equal to the Kripke-Kleene fixpoint $(\mathcal{I}_{\{p,q\}}, \{\})$ of $\mathcal{D}_T$.

Using the correspondences between consistent (stable) fixpoints of $\tilde{\mathcal{D}}_T$ and $\mathcal{D}_T$, we can now proceed to analyze T by looking at $\tilde{\mathcal{D}}_T$ and moreover, because $\tilde{\mathcal{D}}_T$ is by construction stratifiable, we can do so by applying our algebraic stratification results to this operator. Concretely, we can incrementally construct its (stable) fixpoints, using its component operators. Of course, for this result to be useful, we also need to know what these component operators actually are. It turns out that a component on level i corresponds to a theory, that can be constructed from T_i, by replacing certain formulas by their truth value according to a belief pair $(\tilde{U}, \tilde{V}) \in \tilde{\mathcal{B}}_\Sigma|_{\prec i}$. Before showing this for all stratifiable theories, we first consider the following, more restricted class of theories.

Definition 3.25. A theory T is *modally separated* w.r.t. to a partition $(\Sigma_i)_{i\in I}$ of its alphabet iff there exists a corresponding partition $(T_i)_{i\in I}$ of T, such that for each $i \in I$ and $\varphi \in T_i$

- $At_O(\varphi) \subseteq \Sigma_i$,
- for each modal subformula $K\psi$ of φ, either $At(\psi) \subseteq \Sigma_i$ or $At(\psi) \subseteq \bigcup_{j\prec i} \Sigma_j$.

Clearly, all modally separated theories are stratifiable. The fact that each modal subformula of a level T_i of a modally separated theory T contains either only atoms from Σ_i or only atoms from a strictly lower level, makes it easy to construct the components of its $\tilde{\mathcal{D}}_T$-operator. Replacing all modal subformulas of a level T_i which contain only atoms from a strictly lower level $j \prec i$, by their truth-value according to a belief pair $(\tilde{U}, \tilde{V}) \in \mathcal{B}_\Sigma|_{\prec i}$ results in a "conservative theory" T^c, while replacing these subformulas by their truth-value according to $(\tilde{V}, \tilde{U})$ yields a "liberal theory" T^l. The pair $(\mathcal{D}^u_{T^l}, \mathcal{D}^u_{T^c})$ is then precisely the component $(\tilde{\mathcal{D}}_T)_i^{(\tilde{U},\tilde{V})}$ of $\tilde{\mathcal{D}}_T$.

To make this more precise, we inductively define the following transformation $\varphi\langle U, V\rangle_i$ of a formula $\varphi \in T_i$, given a belief pair $(\tilde{U}, \tilde{V}) \in \tilde{\mathcal{B}}_\Sigma|_{\prec i}$:

- $a\langle\tilde{U}, \tilde{V}\rangle_i = a$ for each atom a;
- $(\varphi_1 \wedge \varphi_2)\langle\tilde{U}, \tilde{V}\rangle_i = \varphi_1\langle\tilde{U}, \tilde{V}\rangle_i \wedge \varphi_2\langle\tilde{U}, \tilde{V}\rangle_i$;
- $(\varphi_1 \vee \varphi_2)\langle\tilde{U}, \tilde{V}\rangle_i = \varphi_1\langle\tilde{U}, \tilde{V}\rangle_i \vee \varphi_2\langle\tilde{U}, \tilde{V}\rangle_i$;
- $(\neg\varphi)\langle\tilde{U}, \tilde{V}\rangle_i = \neg(\varphi\langle\tilde{V}, \tilde{U}\rangle_i)$;
- $(K\varphi)\langle\tilde{U}, \tilde{V}\rangle_i = \begin{cases} \mathcal{H}_{\overline{\kappa}(\tilde{U},\tilde{V}),\cdot}(K\varphi) & \text{if } At(\varphi) \subseteq \bigcup_{j\prec i} \Sigma_j; \\ K(\varphi) & \text{if } At(\varphi) \subseteq \Sigma_i. \end{cases}$

Note that this transformation $\varphi\langle\tilde{U}, \tilde{V}\rangle_i$ is identical to the transformation $\varphi\langle P, S\rangle$ defined earlier, except for the fact that in this case, we only replace modal subformulas with atoms from $\bigcup_{j\prec i} \Sigma_j$ and leave modal subformulas with atoms from Σ_i untouched.

Let us now consider a component $(\tilde{\mathcal{D}}_T)_i^{(\tilde{U},\tilde{V})}$ of the $\tilde{\mathcal{D}}_T$-operator of a modally separated theory T. Such a component maps each $(\tilde{P}_i, \tilde{S}_i) \in \tilde{\mathcal{B}}_{\Sigma_i}$ to $\tilde{\mathcal{D}}_T(\tilde{P}, \tilde{S})|_i$, where $(\tilde{P}, \tilde{S})$ is any element of $\tilde{\mathcal{B}}_\Sigma$ that coincides with $(\tilde{U}, \tilde{V})$ on all levels $j \prec i$ and with $(\tilde{P}_i, \tilde{S}_i)$ on level i. As such, in the construction of a new belief pair $(\tilde{P}'_i, \tilde{S}'_i)$, this component will evaluate modal literals that appear in T_i and whose atoms belong to

$\bigcup_{j \prec i} \Sigma_j$, according to either $\overline{\kappa}((\tilde{P}, \tilde{S})|_{\preceq i})|_{\prec i}$ or $\overline{\kappa}((\tilde{S}, \tilde{P})|_{\preceq i})|_{\prec i}$. Now, if $(\tilde{P}_i, \tilde{S}_i)$ is consistent, then $\overline{\kappa}((\tilde{P}, \tilde{S})|_{\preceq i})|_{\prec i} = (\tilde{U}, \tilde{V})$ and $\overline{\kappa}((\tilde{S}, \tilde{P})|_{\preceq i})|_{\prec i} = (\tilde{V}, \tilde{U})$. It follows that we can characterize the behaviour of a component $(\tilde{\mathcal{D}}_T)_i^{(\tilde{U},\tilde{V})}$ on consistent belief pairs as follows:

Proposition 3.22. *Let T be a modally separated theory. Let $i \in I$ and $(\tilde{U}, \tilde{V}) \in \tilde{\mathcal{B}}_\Sigma|_{\prec i}$. For all consistent $(\tilde{P}_i, \tilde{S}_i) \in \tilde{\mathcal{B}}^c_{\Sigma_i}$:*

$$(\tilde{\mathcal{D}}_T)_i^{(\tilde{U},\tilde{V})}(\tilde{P}_i, \tilde{S}_i) = (\mathcal{D}^u_{T_i\langle \tilde{V}, \tilde{U}\rangle_i}(\tilde{S}_i, \tilde{P}_i), \mathcal{D}^u_{T_i\langle \tilde{U}, \tilde{V}\rangle_i}(\tilde{P}_i, \tilde{S}_i)).$$

Now, all that remains to be done is to characterize the components of stratifiable theories which are not modally separated. It turns out that for each stratifiable theory T, there exists a modally separated theory T', which is equivalent to T with respect to evaluation in disconnected possible world structures. To simplify the proof of this statement, we recall that each formula φ can be written in an equivalent form φ' such that each modal subformula of φ' is of the form $K(a_1 \vee \cdots \vee a_m)$, with each a_i an objective literal. This result is well-known for S_5 semantics and can — using the same transformation — be shown to also hold for all semantics considered here[2].

Proposition 3.23. *Let (P, S) be a disconnected element of $\mathcal{B}_\Sigma$. Let $i \in I$, $b_1, \ldots, b_n$ literals with atoms from Σ_i and $c_1, \ldots, c_m$ literals with atoms from $\bigcup_{j \prec i} \Sigma_j$. Then*

$$\mathcal{H}_{(P,S),\cdot}(K(\bigvee_{j=1..n} b_j \vee \bigvee_{j=1..m} c_j)) = \mathcal{H}_{(P,S),\cdot}(K(\bigvee_{j=1..n} b_j) \vee K(\bigvee_{j=1..m} c_j)).$$

Proof. By definition,

$$\mathcal{H}_{(P,S),\cdot}(K(\bigvee_{j=1..n} b_j \vee \bigvee_{j=1..m} c_j)) = \mathbf{t}$$

iff

$$\forall X \in P : \mathcal{H}_{(\cdot,\cdot),X}(\bigvee_{j=1..n} b_j \vee \bigvee_{j=1..m} c_j) = \mathbf{t}.$$

This is equivalent to $\forall X \in P, \mathcal{H}_{(\cdot,\cdot),X}(\bigvee_{j=1..n} b_j) = \mathbf{t}$ or $\mathcal{H}_{(\cdot,\cdot),X}(\bigvee_{j=1..m} c_j) = \mathbf{t}$. Because P is disconnected, it contains all possible combinations $X|_{\cup_{j \prec i}\Sigma_j} \cup Y|_{\Sigma_i} \cup Z|_{\cup_{j \not\preceq i}\Sigma_j}$, with $X, Y, Z \in P$. Therefore the previous statement is in turn equivalent to for each $X, Y \in P, \mathcal{H}_{(\cdot,\cdot),X|_{\Sigma_i}}(\bigvee_{j=1..n} b_j) = \mathbf{t}$ or $\mathcal{H}_{(\cdot,\cdot),Y|_{\cup_{j \prec i}\Sigma_j}}(\bigvee_{j=1..m} c_j)$, which proves the result. □

As previously discussed, given a level T_i of a stratifiable autoepistemic theory T, we can construct an equivalent theory T_i' in which every modal subformula is of the

[2] To show this, it suffices to show that each step of this transformation preserves the value of the evaluation $\mathcal{H}_{(P,S),X}(\varphi)$. For all steps corresponding to properties of (three-valued) propositional logic, this is trivial. The step of transforming a formula $K(K(\varphi))$ to $K(\varphi)$ also trivially satisfies this requirement. All that remains to be shown, therefore, is that $\mathcal{H}_{(P,S),\cdot}(K(\varphi \wedge \psi)) = \mathcal{H}_{(P,S),\cdot}(K(\varphi) \wedge K(\psi))$. By definition, $\mathcal{H}_{(P,S),\cdot}(K(\varphi \wedge \psi)) = \mathbf{t}$ iff $\forall X \in P : \mathcal{H}_{(P,S),X}(\varphi) = \mathbf{t}$ and $\mathcal{H}_{(P,S),X}(\varphi) = \mathbf{t}$, which in turn is equivalent to $\forall X \in P : \mathcal{H}_{(P,S),X}(K(\varphi)) = \mathbf{t}$ and $\forall X \in P : \mathcal{H}_{(P,S),X}(K(\psi)) = \mathbf{t}$.

form $K(a_1 \vee \cdots \vee a_n)$, with the a_i objective literals. Using the above proposition, we can further split every modal subformula of such a T_i' into a part containing only symbols from Σ_i and a part containing only symbols from $\bigcup_{j \prec i} \Sigma_j$, thus creating a modally separated theory T_i'', which is equivalent to T_i with respect to evaluation in pairs of disconnected possible world structures. We will denote this T_i'' as $[T_i]$. In the case of our example $F = \{p \vee \neg Kp;\ K(p \vee q) \vee q\}$, the modally separated theory $[F] = \{p \vee \neg Kp;\ K(p) \vee K(q) \vee q\}$ is equivalent to F with respect to evaluation in disconnected belief pairs. Together with Proposition 3.22, the fact that for all $(\tilde{P}, \tilde{S}) \in \tilde{\mathcal{B}}_\Sigma$ and $X \in \mathcal{W}_\Sigma$, $\mathcal{H}_{\overline{\kappa}(\tilde{P},\tilde{S}),X}(T_i) = \mathcal{H}_{\overline{\kappa}(\tilde{P},\tilde{S}),X}([T_i])$ implies the following characterization of the component operators of $\tilde{\mathcal{D}}_T$.

Proposition 3.24. *Let $i \in I$ and $(\tilde{U}, \tilde{V}) \in \tilde{\mathcal{B}}_\Sigma|_{\prec i}$. For all consistent $(\tilde{P}_i, \tilde{S}_i) \in \tilde{\mathcal{B}}^c_{\Sigma_i}$:*

$$(\tilde{\mathcal{D}}_T)_i^{(\tilde{U},\tilde{V})}(\tilde{P}_i, \tilde{S}_i) = (\mathcal{D}^u_{[T_i]\langle \tilde{V}, \tilde{U}\rangle_i}(\tilde{S}_i, \tilde{P}_i), \mathcal{D}^u_{[T_i]\langle \tilde{U}, \tilde{V}\rangle_i}(\tilde{P}_i, \tilde{S}_i)).$$

Because we already know that we can construct consistent fixpoints of the operator $\mathcal{D}_T$ by incrementally constructing consistent fixpoints of the component operators of $\tilde{\mathcal{D}}_T$, this result now provides the final piece of the puzzle, by showing how these component operators can be derived from the theory T. For $i \in I$ and $(\tilde{U}, \tilde{V}) \in \tilde{\mathcal{B}}_\Sigma|_{\prec i}$, let us define that a belief pair $(\tilde{P}_i, \tilde{S}_i) \in \tilde{\mathcal{B}}_{\Sigma_i}$ is a *stratified partial expansion* of stratum T_i given $(\tilde{U}, \tilde{V})$ if it is a fixpoint of the component $(\tilde{\mathcal{D}}_T)_i^{(\tilde{U},\tilde{V})}$. A belief pair (P, S) is now a consistent partial expansion of T if and only if there exists a $(\tilde{P}, \tilde{S})$, such that $\overline{\kappa}(\tilde{P}, \tilde{S}) = (P, S)$ and, for each $i \in I$, $(\tilde{P}, \tilde{S})(i)$ is a consistent stratified partial expansion of T_i given $(\tilde{P}, \tilde{S})|_{\prec i}$.

Note that if $\tilde{U} = \tilde{V}$, then of course $[T_i]\langle \tilde{V}, \tilde{U}\rangle = [T_i]\langle \tilde{U}, \tilde{V}\rangle$, which means that, on consistent belief pairs, the component $(\tilde{\mathcal{D}}_T)_i^{(\tilde{U},\tilde{V})}$ coincides with the operator $\tilde{\mathcal{D}}_{T'}$ for the theory $T' = [T_i]\langle \tilde{U}, \tilde{V}\rangle_i$. As such, the stratified partial expansions of T_i given some exact pair $(\tilde{U}, \tilde{U})$ are simply the partial expansions of the theory $[T_i]\langle \tilde{U}, \tilde{U}\rangle_i$.

Example 3.7. Let us illustrate this be means of the example F, which we previously partitioned into $F_0 = \{p \vee \neg Kp\}$ and $F_1 = \{K(p \vee q) \vee q\}$. The belief pair $(\{\{\}, \{p\}\}, \{\{p\}\})$ is a consistent partial expansion of F_0, as can be seen from the following equations:

$$Mod(F_0\langle \{\{p\}\},\ \{\{\}, \{p\}\}\rangle) = Mod(p \vee \neg \mathbf{f}) = \{\{\}, \{p\}\};$$
$$Mod(F_0\langle \{\{\}, \{p\}\},\ \{\{p\}\}\rangle) = Mod(p \vee \neg \mathbf{t}) = \{\{p\}\}.$$

For the second level, we need to consider $[F_1] = \{Kp \vee Kq \vee q\}$ and use this to construct two theories F_1^l and F_1^c, to be used in the construction of, respectively, the underestimates and overestimates for Σ_1, that is, F_1^l liberally estimates the truth of Kp, while F_1^c estimates it conservatively:

$$F_1^l = [F_1]\langle \{\{p\}\},\ \{\{\}, \{p\}\}\rangle_1 = \{\mathbf{t} \vee Kq \vee q\};$$
$$F_1^c = [F_1]\langle \{\{\}, \{p\}\},\ \{\{p\}\}\rangle_1 = \{\mathbf{f} \vee Kq \vee q\}.$$

The above proposition now tells us that the component operator $(\tilde{\mathcal{D}}_T)_1^{(\{\{\},\{p\}\},\{\{p\}\})}$ coincides with the operator $(\tilde{\mathcal{D}}^u_{F_1^l}, \tilde{\mathcal{D}}^u_{F_1^c})$ on consistent belief pairs. Therefore, this component has a fixpoint $(\{\{\},\{q\}\},\{\{q\}\})$, as can be seen from the following equations:

$$Mod(F_1^l\langle\{\{q\}\},\ \{\{\},\{q\}\}\rangle) = Mod(\mathbf{t} \vee \mathbf{t} \vee q) = \{\{\},\{q\}\};$$
$$Mod(F_1^c\langle\{\{\},\{q\}\}\rangle,\ \{\{q\}\}) = Mod(\mathbf{f} \vee \mathbf{f} \vee q) = \{\{q\}\}.$$

Now, the set of all $I \cup J$ for which $I \in \{\{\},\{p\}\}$ and $J \in \{\{\},\{q\}\}$ is $\mathcal{I}_{\{p,q\}}$, while the set of all $M \cup N$ for which $M \in \{\{p\}\}$ and $N \in \{\{q\}\}$ is the singleton $\{\{p,q\}\}$. We therefore conclude that the belief pair $(\mathcal{I}_{\{p,q\}}, \{\{p,q\}\})$ is a consistent partial expansion of T.

So far, the above discussion has only considered partial expansions. We can of course also look at other semantics. Let us define a *stratified partial extension* of T_i given $(\tilde{U},\tilde{V})$ as a stable fixpoint of the component $(\tilde{\mathcal{D}}_T)_i^{(\tilde{U},\tilde{V})}$. We also introduce the term *stratified expansion* (or *stratified extension*) to refer to a $\tilde{Q} \in \tilde{\mathcal{W}}_\Sigma$ for which $(\tilde{Q},\tilde{Q})$ is a stratified partial expansion (respectively, stratified partial extension).

Theorem 3.9. *Let T be a stratifiable autoepistemic theory. A belief pair (P,S) of $\mathcal{B}_\Sigma$ is a consistent partial expansion (respectively, consistent partial extension) of T iff there exists a $(\tilde{P},\tilde{S}) \in \tilde{\mathcal{B}}_\Sigma$, such that $\overline{\kappa}(\tilde{P},\tilde{S}) = (P,S)$ and for all $i \in I$, $(\tilde{P},\tilde{S})(i)$ is a consistent stratified partial expansion (consistent stratified partial extension) of T_i given $(\tilde{P},\tilde{S})|_{\prec i}$. A possible world structure $Q \in \mathcal{W}_\Sigma$ is a consistent expansion (respectively, consistent extension) of T iff there exists a $\tilde{Q} \in \tilde{\mathcal{W}}_\Sigma$, such that $\kappa(\tilde{Q}) = Q$ and for all $i \in I$, $\tilde{Q}(i)$ is a consistent expansion (consistent extension) of $T_i\langle(\tilde{Q},\tilde{Q})|_{\prec i}\rangle_i$.*

For permaconsistent theories, we can draw stronger conclusions. Let us call the least stratified partial expansion (respectively, least stratified partial extension) the *stratified Kripke-Kleene model* (*stratified well-founded model*). We then have the following result.

Theorem 3.10. *Let T be a stratifiable autoepistemic theory. If T is also permaconsistent, then (P,S) is a partial expansion, partial extension, Kripke-Kleene model or well-founded model of T iff there exists a $(\tilde{P},\tilde{S}) \in \tilde{\mathcal{B}}_\Sigma$, such that $\overline{\kappa}(\tilde{P},\tilde{S}) = (P,S)$ and for all $i \in I$, $(\tilde{P},\tilde{S})(i)$ is, respectively, a stratified partial expansion, stratified partial extension, stratified Kripke-Kleene model or stratified well-founded model of T_i given $(\tilde{P},\tilde{S})|_{\prec i}$. A possible world structure $Q \in \mathcal{W}_\Sigma$ is an expansion (extension, respectively) of T iff there exists a $\tilde{Q} \in \tilde{\mathcal{W}}_\Sigma$, such that $\kappa(\tilde{Q}) = Q$ and for all $i \in I$, $\tilde{Q}(i)$ is an expansion (extension) of $T_i\langle(\tilde{Q},\tilde{Q})|_{\prec i}\rangle_i$.*

3.4.1 Related work

In (Gelfond and Przymusinska 1992) and (Niemelä and Rintanen 1994), it was shown that certain permaconsistent and modally separated autoepistemic theories can be split under the semantics of expansions. We have both extended these results to other semantics for this logic and to a larger class of theories.

To give some intuition about the kind of theories our result can deal with, but previous work cannot, we will consider the following example (from (Etherington 1988)):

Suppose we would like to express that we suspect a certain person of murder if we know he had a motive and if it is possible that this person is a suspect and that he is guilty. This naturally leads to following formula:

$$Kmotive \wedge \neg K(\neg suspect \vee \neg guilty) \rightarrow suspect.$$

This formula is not modally separated w.r.t. the partition

$$\Sigma_0 = \{guilty, motive\}, \Sigma_1 = \{suspect\}$$

and, therefore, falls outside the scope of Gelfond et al.'s theorem. Our result, however, does cover this example and allows it to be split w.r.t. this partition. As we will discuss in the next section on default logic, there exists an important class of default expressions, called *semi-normal defaults*, which typically give rise to such statements.

3.5 Application to default logic

We recall that a default theory is a pair $\langle D, W\rangle$, where W is a set of propositional formulas and D is a set of defaults of the form:

$$\frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma}$$

We defined the semantics of such a theory by a transformation m to autoepistemic logic, which maps each default d of the above form to:

$$(K\alpha \wedge \neg K\neg\beta_1 \wedge \cdots \wedge \neg K\neg\beta_n \Rightarrow \gamma).$$

The formula γ is called the consequence $cons(d)$ of d.

We begin by defining the concept of a dependency relation for a default theory.

Definition 3.26. Let $\langle D, W\rangle$ be a default theory over an alphabet Σ. A binary relation $\rightsquigarrow$ on Σ is a dependency relation for this theory if it satisfies the following conditions:

- For each default d: if p is an atom appearing anywhere in d and q is an atom appearing in $cons(d)$, then $p \rightsquigarrow q$;
- For each $w \in W$, if atoms p, q appear in w, then $p \rightsquigarrow q$.

Again, we can define a default theory $\langle D, W\rangle$ to be stratifiable with respect to a partition $(\Sigma_i)_{i\in I}$ of its alphabet iff this partition respects a dependency relation for $\langle D, W\rangle$. It can easily be seen that, for such a stratifiable theory, there must exist a corresponding partition $\langle D_i, W_i\rangle_{i\in I}$ of $\langle D, W\rangle$ such that:

- For each default d: if an atom of $cons(d)$ is in Σ_i, then $d \in D_i$,
- For each default d: all atoms of d are in $\bigcup_{j\preceq i} \Sigma_j$.
- For each $w \in W$, if w contains an atom $p \in \Sigma_i$, then $w \in W_i$.

It can now easily be seen that a dependency relation for a default theory is also a dependency relation for the corresponding autoepistemic theory and that each level T_i of $m(D, W)$ is precisely $m(D_i, W_i)$.

By the results of the previous section, this now immediately implies the following result.

Theorem 3.11. *Let $\langle D, W\rangle$ be a stratifiable default theory. A belief pair (P, S) is a consistent partial expansion (respectively, consistent partial extension) of $\langle D, W\rangle$ iff there exists a $(\tilde{P}, \tilde{S})$ such that $\overline{\kappa}(\tilde{P}, \tilde{S}) = (P, S)$ and for each $i \in I$, $(\tilde{P}, \tilde{S})(i)$ is a consistent stratified partial expansion (consistent stratified partial extension) of the autoepistemic theory $m(D_i, W_i)$ given $(\tilde{P}, \tilde{S})|_{\prec i}$. A possible world structure Q is a consistent expansion (respectively, consistent extension) of $\langle D, W\rangle$ if there exists a $\tilde{Q}$ such that $\kappa(\tilde{Q}) = Q$ and for each $i \in I$, $\tilde{Q}(i)$ is a consistent stratified expansion (consistent stratified extension) of the autoepistemic theory $m(D_i, W_i)$ given $(\tilde{Q}, \tilde{Q})|_{\prec i}$.*

Consistent stratified partial expansions and consistent stratified partial extensions of the autoepistemic theory $T_i = m(D_i, W_i)$ can be constructed using the component theories $[T_i]\langle(\tilde{S}, \tilde{P})|_{\prec i}\rangle_i$ and $[T_i]\langle(\tilde{P}, \tilde{S})|_{\prec i}\rangle_i$. Because no such component theory can contain a nested occurrence of a modal literal, standard transformations for propositional logic can be used to bring each of its formulas into the form:

$$\neg(K\alpha_1 \wedge K\alpha_2 \wedge \cdots \wedge K\alpha_m) \vee K\beta_1 \vee \cdots \vee K\beta_n \vee \gamma,$$

where the α_i, β_j and γ are all propositional formulas. Because $K\alpha_1 \wedge \cdots \wedge K\alpha_m$ is equivalent to $K(\alpha_1 \wedge \cdots \wedge \alpha_m)$, this suffices to show that each such formula can be transformed back into a default. As such, it would be possible to reformulate the above theorem entirely in terms of default logic.

We now investigate a specific class of theories, for which it is particularly useful to restate our result in this manner. Let us call a default theory $\langle D, W\rangle$ *modally separated* if the autoepistemic theory $m(D, W)$ is modally separated or, equivalently, if for every default $d \in D_i$ of form (3.5), it is the case that no formula $\varphi \in \{\alpha, \beta_1, \ldots, \beta_n\}$ contains both an atom $p \in \Sigma_i$ and $q \in \bigcup_{j \prec i} \Sigma_j$. Given a stratum $\langle D_i, W_i\rangle$ of such a theory and a belief pair $(\tilde{U}, \tilde{V}) \in \tilde{\mathcal{B}}_\Sigma|_{\prec i}$, we define $D_i\langle\tilde{U}, \tilde{V}\rangle_i$ as the set of defaults d' that result from replacing, in every $d \in D_i$, all formulas $\varphi \in \{\alpha, \beta_1, \ldots, \beta_n\}$ whose atoms belong to alphabet $\bigcup_{j \prec i} \Sigma_j$, by their truth value according to $(\tilde{U}, \tilde{V})$. Now, we can characterize the autoepistemic component theory $[T_i]\langle\tilde{U}, \tilde{V}\rangle_i$ as simply being $m(D_i\langle\tilde{U}, \tilde{V}\rangle_i, W_i)$. Therefore, Theorem 3.11 implies that a possible world structure P is a consistent extension of a stratifiable default theory $\langle D, W\rangle$ if and only if there exists a $\tilde{P}$ such that $\kappa(\tilde{P}) = P$ and, for each $i \in I$, $\tilde{P}(i)$ is a consistent extension of the default theory $\langle D_i\langle(\tilde{P}, \tilde{P})|_{\prec i}\rangle_i, W_i\rangle$.

3.5.1 Related work

(Turner 1996) proved splitting theorems for default logic under the semantics of extensions. We have extended these results to the semantics of partial extensions, (partial) expansions and the Kripke-Kleene and well-founded semantics. Moreover, Turner's

results only apply to modally separated default theories. Our results therefore not only generalize them to other semantics, but also to a larger class of theories.

A typical example of a default which is not modally separated but which can be split using our results, is the example from (Etherington 1988) concerning murder suspects. This can be formalized by the following default:

$$\frac{motive : suspect \wedge guilty}{suspect}.$$

In the previous section, we already presented the autoepistemic formula resulting from applying the Konolige transformation to this default and showed that it was not modally stratified w.r.t. the partition:

$$\Sigma_0 = \{guilty, motive\}, \Sigma_1 = \{suspect\}.$$

Therefore, Turner's theorem does not apply in this case, but our results do.

Defaults such as these are typical examples of so-called *semi-normal defaults*, i.e., defaults of the form:

$$\frac{\alpha : \beta}{\gamma}$$

where β implies γ. This typically occurs because there is some formula δ, such that $\beta = \gamma \wedge \delta$. In such cases, the Konolige transformation will contain a formula $K(\neg\gamma \vee \neg\delta)$ and such defaults can therefore only be modally separated w.r.t. stratifications in which all atoms from both γ and δ belong to the same stratum. Our results, however, also allows stratifications in which (all or some) atoms from δ belong to a strictly lower stratum than the atoms from γ.

3.6 Conclusions

Stratification is, both theoretically and practically, an important concept in knowledge representation. We have studied this issue at a general, algebraic level by investigating stratification of operators in the setting of approximation theory. This gave us a small but useful set of theorems, which enabled us to easily and uniformly prove splitting results for all fixpoint semantics of logic programs, autoepistemic logic and default logic, thus generalizing existing results. As such, the importance of the work presented in this chapter is threefold. First, there are our concrete, applied results for logic programs, autoepistemic logic and default logic themselves. Second, there is the general, algebraic framework for the study of stratification, which can be applied to every formalism with a fixpoint semantics. Finally, on a more abstract level, our work also offers greater insight into the principles underlying various existing stratification results, as we are able to "look beyond" purely syntactical properties of a certain formalism.

Chapter 4

Predicate introduction

4.1 Introduction

In this chapter, we study the problem of "predicate introduction". To introduce this topic, let us consider logic programming. In this context, predicate introduction refers to a transformation that introduces a new predicate in order to simplify the expressions in the bodies of certain rules. To motivate our interest in this, we consider a simplified version of a program that occurs in (Balduccini and Gelfond 2003). In this paper, a logic program (under the stable model semantics) is constructed to capture the meaning of theories in the action language $\mathcal{AL}$. In particular, *static causal laws* of the following form are considered: "P is caused if $P_1, \ldots, P_N$". Here, P, P_1,..., P_N are propositional symbols. In its logic programming translation, such a causal law R is represented by the following set of facts: $\{Head(R, P), Prec(R, 1, P_1), \ldots, Prec(R, N, P_N), NbOfPrec(R, N)\}$.

Now, the meaning in $\mathcal{AL}$ of such a law is that whenever all of $P_1, \ldots, P_N$ hold, then so must P. Using the predicate $Holds/1$ to describe which propositions hold, this can be captured by the following rule:

$$\forall p\ Holds(p) \leftarrow (\exists r\ Head(r, p) \wedge \forall i \forall q\ Prec(r, i, q) \Rightarrow Holds(q)). \tag{4.1}$$

This rule contains universal quantifiers in its body. Even though, as we have seen in Section 2.2, it is possible to define both stable and well-founded semantics for such programs, current model generation systems such as ASSAT, SModels or DLV cannot handle this kind of rules. Therefore, we would like to eliminate this quantifier. The well-known Lloyd-Topor transformation (Lloyd and Topor 1984) suggests introducing a new predicate, $BodyNotSat/1$, to represent the negation of the subformula $\varphi = (\forall i \forall q\ Prec(r, i, q) \Rightarrow Holds(q))$. Since $\neg\varphi = (\exists i \exists q\ Prec(r, i, q) \wedge \neg Holds(q))$, we would then get:

$$\begin{aligned} \forall p, r\ Holds(p) &\leftarrow Head(r, p) \wedge \neg BodyNotSat(r). \\ \forall r, i, q\ BodyNotSat(r) &\leftarrow Prec(r, i, q) \wedge \neg Holds(q). \end{aligned} \tag{4.2}$$

This transformation preserves equivalence under the (two-valued) completion semantics (Lloyd and Topor 1984). However, for stable or well-founded semantics, this is not

the case. For instance, consider the $\mathcal{AL}$ theory $\mathcal{A} = \{P$ is caused if Q; Q is caused if $P\}$. In the original translation (4.1), neither P nor Q holds; in the second version (4.2), however, we obtain (ignoring the $Head/2$ and $Prec/3$ atoms for clarity):

$$\begin{aligned} Holds(P) &\leftarrow \neg BodyNotSat(R_1). \\ BodyNotSat(R_1) &\leftarrow \neg Holds(Q). \\ Holds(Q) &\leftarrow \neg BodyNotSat(R_2). \\ BodyNotSat(R_2) &\leftarrow \neg Holds(P). \end{aligned} \tag{4.3}$$

Under the stable semantics, this program has two models: $\{Holds(P), Holds(Q)\}$ and $\{BodyNotSat(R_1), BodyNotSat(R_2)\}$. As such, even though it might look reasonable at first, the Lloyd-Topor transformation does not preserve stable (or well-founded) models in this case.

Predicate introduction under the well-founded semantics was considered by Van Gelder (Van Gelder 1993). That paper, however, imposes strong restrictions on how newly introduced predicates can be defined. In particular, recursive definitions of such a new predicate are not allowed. However, the ability to introduce recursively defined new predicates can be very useful; indeed, it is precisely in this way that (Balduccini and Gelfond 2003) manages to eliminate the universal quantifier in (4.1).

Example 4.1 (Adapted from (Balduccini and Gelfond 2003)). In order to replace the universally quantified subformula $\forall i \forall q\ Prec(r, i, q) \Rightarrow Holds(q)$ of (4.1), we introduce a new predicate $AllPrecHold(r)$, resulting in:

$$\forall r, p\ Holds(p) \leftarrow Head(r, p) \wedge AllPrecHold(r). \tag{4.4}$$

This predicate is then defined in terms of another new predicate, $AllFrom(r, i)$, that means that the preconditions $i, i+1, \ldots, n$ of a rule r with n preconditions are satisfied. We then define this predicate by the following recursion:

$$\begin{aligned} &\forall r\ AllPrecHold(r) \leftarrow AllFrom(r, 1). \\ &\forall r, n\ AllFrom(r, n) \leftarrow \exists q\ Prec(r, n, q) \wedge Holds(q) \wedge AllFrom(r, n+1). \\ &\forall r, n\ AllFrom(r, n) \leftarrow \exists q\ Prec(r, n, q) \wedge Holds(q) \wedge NbOfPrec(r, n). \end{aligned} \tag{4.5}$$

In this chapter, we prove a generalization of Van Gelder's result, that shows that this translation is indeed equivalence preserving.

So far, we have motivated our interest in predicate introduction by looking at logic programming. However, this same principle is if course also useful in other knowledge representation languages. Again, the framework of approximation theory will allow us to study the concept of predicate introduction in a general and uniform way, independent of any specific logic. Concretely, we will, in this setting, define the abstract, algebraic notion of a *fixpoint extension* of an operator, which captures predicate introduction at the level of approximation theory. The central result of this chapter is then a theorem that relates the fixpoints of such a fixpoint extension (which, intuitively, correspond to models of the transformed theory) to those of the original operator it extends (which correspond to models of the original theory).

By instantiating this algebraic theorem to the case of logic programming, we will be able to prove certain equivalences under the well-founded and stable model semantics, which generalize the aforementioned result by Van Gelder (Van Gelder 1993), by also allowing recursively defined new predicates. This has some interesting applications, including a general way of eliminating universal quantifiers. As such, we offer an alternative for the corresponding step from the Lloyd-Topor transformation, which is only valid under completion semantics.

We also use the same theorem to prove a result for autoepistemic logic. Here, we study transformations that introduce new propositions to reduce the nesting level of the modal operator K. For instance, in the formula

$$\neg K(r \vee \neg Ks)$$

the K operator is nested to depth 2. By introducing a new proposition p to replace Ks, we can transform this formula to $\neg K(r \vee \neg p)$, with nesting depth 1. The new proposition p can then be 'defined' by the formula $Ks \Rightarrow p$. We will show that, on an algebraic level, what happens in this case is precisely the same as what happens when we perform predicate introduction in a logic program. As such, we will be able to prove that this transformation is equivalence preserving under a number of different semantics for autoepistemic logic.

Unlike Chapter 3, this chapter will not discuss the application of our algebraic result to default logic. Indeed, as we also saw in 3.5, there is little to be learned from this, due the very close relation between default logic and autoepistemic logic.

4.2 Predicate introduction in approximation theory

We want to study the following transformation. We start out with a rule set Δ in some alphabet Σ and then introduce an additional alphabet σ of new symbols, e.g., the two predicates $AllPrecHold$ and $AllFrom$ from the example in the introduction, that are defined by some additional rules δ. We then use these new predicates to form a new definition Δ' over alphabet $\Sigma \cup \sigma$. In order to study such transformations in an algebraic setting, we will assume two complete lattices $\langle L_1, \leq_1 \rangle$ and $\langle L_2, \leq_2 \rangle$. Here, L_1 can be thought of as consisting of the interpretations for the original alphabet Σ, while L_2 represents the interpretations for the additional new alphabet σ. We will need to prove a result concerning the stable and well-founded models of Δ', which means that we will need to work with pairs of interpretations of $\Sigma \cup \sigma$. As such, we consider the square $(L_1 \times L_2)^2$ of the product $L_1 \times L_2$, which, as we have seen, is isomorphic to the product $L_1^2 \times L_2^2$ of the squares of these lattices. We will denote pairs $P = ((x, y), (u, v))$ of this latter Cartesian product as $\left(\begin{smallmatrix} x & y \\ u & v \end{smallmatrix}\right)$, where $(x, y) \in L_1^2$ and $(u, v) \in L_2^2$. For consistency with this matrix-like notation, we will also write pairs $(x, y) \in L_1^2$ as $(x\ y)$ and pairs (x, u) in $L_1 \times L_2$ as $\binom{x}{u}$. We define the following projection functions: by $[P|$ we denote the pair $\binom{x}{u}$, by $|P]$ the pair $\binom{y}{v}$, by $\lceil P]$ the pair $(x\ y)$, by $\lfloor P\rfloor$ the pair $(u\ v)$, by $\lfloor P|$ the element u, by $\lceil P|$ the element x, by $|P]$ the element y, and by $|P\rfloor$ the element v.

Now, we want to prove a relation between the stable and well-founded fixpoints of the operator $\mathcal{T}_\Delta$ of the original definition Δ and those of the new operator $\mathcal{T}_{\Delta'}$.

Algebraically, we consider an approximation A on the square L_1^2 of the original lattice L_1 and an approximation B on the extended lattice $L_1^2 \times L_2^2$. We now impose some conditions to ensure a correspondence between the stable fixpoints of A and B.

The main idea behind these conditions is the following. By introducing a new predicate into our original definition Δ, we have added an additional "indirection". For instance, in the original version Δ of our example, we had the formula

$$\forall i, q\ Prec(r, i, q) \Rightarrow Holds(q),$$

that could be evaluated in order to check whether all preconditions q of rule r were satisfied. This could be done by the $\mathcal{T}_\Delta$-operator in a single step. In our new definition Δ', however, every application of $\mathcal{T}_{\Delta'}$ only checks whether a single precondition is satisfied. Intuitively, to match the effect of a single application of $\mathcal{T}_\Delta$ to some pair (X, Y) of interpretations of the alphabet of Δ, we have to iterate $\mathcal{T}_{\Delta'}$ long enough for the truth assignments of (X, Y) to propagate throughout all of the new symbols of Δ'. Nevertheless, the end result of this iteration of $\mathcal{T}_{\Delta'}$ should coincide with the result of the single application of $\mathcal{T}_\Delta$. We need some more notation to formalize this intuition.

Given the operator B on $L_1^2 \times L_2^2$ and a pair $(x\ y) \in L_1^2$, we define the operator $B^{(x\ y)}$ on L_2^2 as $\lambda(u\ v).\lfloor B(\begin{smallmatrix} x & y \\ u & v \end{smallmatrix})\rfloor$. Conversely, given a pair $(u\ v) \in L_2^2$, we define the operator $B_{(u\ v)}$ on L_1^2 as $\lambda(x\ y).\lceil B(\begin{smallmatrix} x & y \\ u & v \end{smallmatrix})\rceil$. Intuitively, the idea is that one application of the operator $B^{(x\ y)}$ will correspond to the act of checking a single precondition in our example. Now, an important property is that the rule set δ defining the new predicates contains only *positive* recursion, i.e., δ is a monotone definition. In our algebraic setting, this leads to a property called *part-to-part monotonicity*, which states that each operator $B^{(x\ y)}$ is monotone.

By itself, however, this form of monotonicity will not suffice. Intuitively, the reason for this is that we should make sure that our transformation does not introduce "too much" non-monotonicity. In the context of logic programming, this means that we should not introduce an additional cycle over negation. Consider, for instance, the singleton rule set $\{P \leftarrow \neg\neg P\}$. The transformation of this rule set into $\{P \leftarrow \neg N, N \leftarrow \neg P\}$ (i.e., replacing $\neg P$ by a new atom N, defined as $N \leftarrow \neg P$) would clearly not preserve stable or well-founded semantics. To avoid this case, we need to make sure that we either replace a positively occurring formula, e.g., replace P in $\{P \leftarrow \neg\neg P\}$, or that our new atom depends only positively on the original atoms. This last case would cover, for instance, the transformation of $\{P \leftarrow \neg P\}$ into $\{P \leftarrow \neg N, N \leftarrow P\}$. In our algebraic setting, we introduce two different ways of strengthening the aforementioned part-to-part monotonicity. The first, called part-to-whole monotonicity, states that both the new and the old predicates depend positively on the new predicates; this covers, e.g., the replacement of P in $\{P \leftarrow \neg\neg P\}$. The second is called whole-to-part monotonicity and states that the new predicates should depend positively on both the new and the old predicates; this covers the transformation of $\{P \leftarrow \neg P\}$ to $\{P \leftarrow \neg N, N \leftarrow P\}$. Note that none of these cases cover the forbidden transformation from $\{P \leftarrow \neg\neg P\}$ to $\{P \leftarrow \neg N, N \leftarrow \neg P\}$. In summary, we get the following algebraic definitions.

Definition 4.1. Let B be an approximation on $L_1^2 \times L_2^2$.

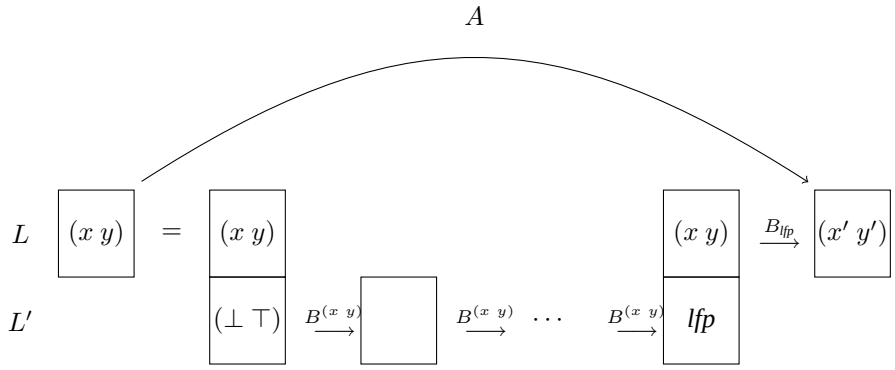


Figure 4.1: B is a predicate extension of A if $B_{\mathit{lfp}(B^{(x\ y)})}(x\ y) = A(x\ y)$.

- B is *part-to-part monotone* iff for each $(x\ y) \in L_1^2$ and $(u\ v) \leq (u'\ v') \in L_2^2$,

$$\lfloor B(\begin{smallmatrix} x & y \\ u & v \end{smallmatrix})\rfloor \leq \lfloor B(\begin{smallmatrix} x & y \\ u' & v' \end{smallmatrix})\rfloor;$$

- B is *part-to-whole monotone* iff for each $(x\ y) \in L_1^2$ and $(u\ v) \leq (u'\ v') \in L_2^2$,

$$B(\begin{smallmatrix} x & y \\ u & v \end{smallmatrix}) \leq B(\begin{smallmatrix} x & y \\ u' & v' \end{smallmatrix});$$

- B is *whole-to-part monotone* iff for all $(\begin{smallmatrix} x & y \\ u & v \end{smallmatrix}) \leq (\begin{smallmatrix} x' & y' \\ u' & v' \end{smallmatrix}) \in L_1^2 \times L_2^2$,

$$\left\lfloor B(\begin{smallmatrix} x & y \\ u & v \end{smallmatrix})\right\rfloor \leq \left\lfloor B(\begin{smallmatrix} x' & y' \\ u' & v' \end{smallmatrix})\right\rfloor.$$

It is easy to see that both part-to-whole and whole-to-part monotonicity imply part-to-part monotonicity. Therefore, if any of these three properties is satisfied, then every operator $B^{(x\ y)}$ will be monotone w.r.t. the product order $\leq$ and, as such, have a $\leq$-least fixpoint $\mathit{lfp}(B^{(x\ y)})$. If we are extending a definition Δ with some new predicates, defined by a monotone rule set δ, it is now precisely this least fixpoint that will tell us what we can obtain by iteratively applying the rules of δ. As explained above, once this derivation using the rules of δ has reached its limit, the operator B should behave as A does on L_1^2. We formalize this in the following definition, which is illustrated in Figure 4.1.

Definition 4.2 (Fixpoint extension). Let B be an approximation on $L_1^2 \times L_2^2$ and A an approximation on L_1^2. B is a *fixpoint extension* of A iff it is part-to-part monotone and, for all $x, y \in L_1$, $B_{\mathit{lfp}_{\leq}(B^{(x\ y)})}(x\ y) = A(x\ y)$.

Our goal is now to prove a correspondence between fixpoints of an approximation A and a fixpoint extension B of A. We begin by making the trivial observation that any fixpoint $(x\ y)$ of A can be extended to a fixpoint $(\begin{smallmatrix} x & y \\ u & v \end{smallmatrix})$ of B, by choosing u and v in an appropriate way.

Theorem 4.1. *Let B be a fixpoint extension of A. A pair $(x\ y) \in L_1^2$ is a fixpoint of A iff $\binom{x\ \ y}{\mathit{lfp}(B^{(x\ y)})}$ is a fixpoint of B.*

This theorem shows that we can find the fixpoints of A by first constructing the fixpoints of B and then checking for which of these fixpoints $\binom{x\ \ y}{u\ \ v}$ it is the case that $(u\ v) = \mathit{lfp}(B^{(x\ y)})$. The following example demonstrates that it is indeed necessary to check this additional condition, because fixpoints of B for which it does not hold might not correspond to fixpoints of A.

Example 4.2. Consider the following rule set:

$$\Delta = \left\{ \begin{array}{l} P \leftarrow Q. \\ Q \leftarrow \mathbf{false}. \end{array} \right\}.$$

Let us try to replace Q in the first rule by R, defined by:

$$\delta = \left\{ \begin{array}{l} R \leftarrow R. \\ R \leftarrow Q. \end{array} \right\}.$$

The resulting rule set Δ' is then of course:

$$\Delta' = \left\{ \begin{array}{l} P \leftarrow R. \\ Q \leftarrow \mathbf{false}. \\ R \leftarrow R. \\ R \leftarrow Q. \end{array} \right\}.$$

As will become clear in Section 4.3, where we discuss predicate introduction for rule sets, the operator $\mathcal{T}_{\Delta'}$ is a fixpoint extension of the operator $\mathcal{T}_{\Delta}$. However, $\mathcal{T}_{\Delta'}$ has a fixpoint in which P holds (namely, the pair $(\{P, R\}, \{P, R\})$), while $\mathcal{T}_{\Delta}$ does not. The reason for the discrepancy is, of course, the rule $R \leftarrow R$ in δ. This positive recursion has the effect that—at least under completion semantics—R might become true, even when Q is false.

This example demonstrates that, given some fixpoint $\binom{x\ \ y}{u\ \ v}$ of B, it is indeed necessary to first check whether $(u\ v) = \mathit{lfp}(B^{(x\ y)})$, before concluding that $(x\ y)$ is a fixpoint of A. Of course, if B happens to be such that, for some reason, we can always be sure that this condition is satisfied, then we can safely ignore it. Indeed, in this case, Theorem 4.1 tells us that the fixpoints $fp(A)$ coincide precisely with the set $\lceil fp(B) \rceil$ of all restrictions to L_1 of fixpoints of B, which implies that also $\mathit{lfp}(A) = \lceil \mathit{lfp}(B) \rceil$. A sufficient condition for this is that every operator $B^{(x\ y)}$ has only a single fixpoint; indeed, since for each fixpoint $\binom{x\ \ y}{u\ \ v}$ of B, $(u\ v)$ is obviously a fixpoint of $B^{(x\ u)}$, the equality $(u\ v) = \mathit{lfp}(B^{(x\ y)})$ then immediately follows. An interesting special case of this is when every $B^{(x\ y)}$ is a constant operator. In this case, we will call the operator B *part-to-part constant*. In logic programming, we can get a part-to-part constant operator by simply disallowing recursion in the rule set defining our new predicates. As

such, this result directly applies to, for instance, any transformation where we replace a subformula $\varphi(\mathbf{x})$ of the body of some rule by a new atom $P(\mathbf{x})$, which we then define $\forall \mathbf{x}\ P(\mathbf{x}) \leftarrow \varphi(\mathbf{x})$. For such transformations, we will therefore get a correspondence between the completion and Kripke-Kleene semantics.

So far, we have considered the relation between fixpoints of A and B, and showed that, in particular, every fixpoint of A has a corresponding fixpoint of B. We now turn our attention to the stable fixpoints of these operators. We first prove that if a fixpoint of A happens to be stable, then the corresponding fixpoint of B will be stable as well.

Theorem 4.2. *Let B be a fixpoint extension of A. If $(x\ y)$ is a fixpoint of the stable operator $\mathcal{C}_A$ of A, then $\begin{pmatrix} x & y \\ \mathit{lfp}(B^{(x\ y)}) \end{pmatrix}$ is a fixpoint of the stable operator $\mathcal{C}_B$ of B.*

To prove this theorem, we first study some more properties of B and $\mathit{lfp}(B^{(x\ y)})$. The operators $B^{(x\ y)}$ are quite special, in the sense that they are both $\leq_p$ and $\leq$-monotone. It can easily be shown that, in general, for any such operator O on a lattice L^2, the first and second component of O are completely independent.

Proposition 4.1. *Let O be an operator on a lattice L^2, such that O is both $\leq$-monotone and $\leq_p$-monotone. For every $(a\ b) \in L^2$, $[O(a\ b)| = [O(a\ \top)|$ and $|O(a\ b)] = |O(\top\ b)]$.*

Proof. Let $a, b \in L$. Because $(a\ b) \leq (a\ \top)$, we have that $O(a\ b) \leq O(a\ \top)$ and, specifically, $[O(a\ b)| \leq [O(a\ \top)|$. Because $(a\ b) \geq_p (a\ \top)$, we have that $O(a\ b) \geq_p O(a\ \top)$ and, specifically, $[O(a\ b)| \geq [O(a\ \top)|$. Therefore, $[O(a\ b)| = [O(a\ \top)|$. Similarly, from $(a\ b) \leq (\top\ b)$ and $(a\ b) \leq_p (\top\ b)$, it follows that, respectively, $|O(a\ b)] \leq |O(\top\ b)]$ and $|O(a\ b)] \geq |O(\top\ b)]$, which proves the result. □

Let us denote by $B_1^{(x\ y)}$ and $B_2^{(x\ y)}$ the operators on L_2, that map, respectively, any $u \in L_2$ to $[B^{(x\ y)}(u\ \top)|$ and any $v \in L_2$ to $|B^{(x\ y)}(\top\ v)]$, i.e., for all $u \in L_2$, $B_1^{(x\ y)}(u) = [B^{(x\ y)}(u\ \top)| = \lfloor B(\begin{smallmatrix} x & y \\ u & \top \end{smallmatrix})|$ and, similarly, for all $v \in L_2$ $B_2^{(x\ y)}(v) = |B^{(x\ y)}(\top\ v)] = |B(\begin{smallmatrix} x & y \\ \top & v \end{smallmatrix})\rfloor$. Clearly, these two operators are monotone.

Because each $B^{(x\ y)}$ is both $\leq$-monotone (since B is part-to-part monotone) and $\leq_p$-monotone (since B is an approximation), Proposition 4.1 now implies the following result.

Proposition 4.2. *Let B be part-to-part monotone. Then for all $x, y \in L_1$ and $u, v \in L_2$,*

$$B^{(x\ y)}(u\ v) = (B_1^{(x\ y)}(u)\ B_2^{(x\ y)}(v)).$$

It follows directly from this proposition that $\mathit{lfp}(B^{(x\ y)}) = (\mathit{lfp}(B_1^{(x\ y)})\ \mathit{lfp}(B_2^{(x\ y)}))$. We can now use this result to show that extending a pair $(x\ y) \in L_1^2$ with $\mathit{lfp}(B^{(x\ y)})$ preserves the precision order.

Proposition 4.3. *Let B be part-to-part monotone. For all $x, x', y, y' \in L_1$,*

$$(x\ y) \leq_p (x'\ y') \text{ iff } \begin{pmatrix} x & y \\ \mathit{lfp}(B^{(x\ y)}) \end{pmatrix} \leq_p \begin{pmatrix} x' & y' \\ \mathit{lfp}(B^{(x'\ y')}) \end{pmatrix}.$$

Proof. It is clear that the right hand side of this equivalence directly implies the left. Let x, x', y, y' be as above and let $(u\ v) = \mathit{lfp}(B^{(x\ y)})$ and $(u'\ v') = \mathit{lfp}(B^{(x'\ y')})$. We have to show that $(u\ v) \leq_p (u'\ v')$. We first show that $u \leq u'$. By Proposition 4.2, $u = \mathit{lfp}(B_1^{(x\ y)})$. Because this implies that u is also the least prefixpoint of $B_1^{(x\ y)}$, it now suffices to show that u' is also a prefixpoint of $B_1^{(x\ y)}$, i.e., that $u' \geq B_1^{(x\ y)}(u')$. Because $\binom{x'\ \ y'}{u'\ \ \top} \geq_p \binom{x\ \ y}{u'\ \ \top}$, we have that $u' = \left\lfloor B\binom{x'\ \ y'}{u'\ \ \top}\right\rfloor \geq \left\lfloor B\binom{x\ \ y}{u'\ \ \top}\right\rfloor = B_1^{(x\ y)}(u')$. The fact that $v \geq v'$ can be shown in a similar way, by proving that v is a prefixpoint of the operator $B_2^{(x'\ y')}$, of which v' is the least prefixpoint. □

The stable operator $\mathcal{C}_B$ of an approximation B is defined in terms of its downward and upward stable operators $C_B^{\downarrow}$ and $C_B^{\uparrow}$. We show the following relation between these operators and the operators $B_1^{(x\ y)}$ and $B_2^{(x\ y)}$.

Proposition 4.4. *Let B be a part-to-part monotone approximation on $L_1^2 \times L_2^2$. If $\binom{x}{u} = C_B^{\downarrow}\binom{y}{v}$, then $u = \mathit{lfp}(B_1^{(x\ y)})$. If $\binom{y}{v} = C_B^{\uparrow}\binom{x}{u}$, then $v = \mathit{lfp}(B_2^{(x\ y)})$.*

Proof. Let B be as above. We only prove the first implication; the proof of the second one is analogous. Let $\binom{x}{u} = C_B^{\downarrow}\binom{y}{v}$ and let $u' = \mathit{lfp}(B_1^{(x\ y)})$. We will show that $u = u'$. We start by showing that $u' \leq u$. By definition of $C_B^{\downarrow}$, $\binom{x}{u} = \left\lceil B\binom{x\ \ y}{u\ \ v}\right\rceil$. In particular, $u = B_1^{(x\ y)}(u)$, i.e., u is a fixpoint of $B_1^{(x\ y)}$. Because u' was chosen to be the least fixpoint of this operator, $u' \leq u$.

Now, we prove that also $u \leq u'$. We do this by constructing an element $x' \in L_1$, such that $\binom{x'}{u'}$ is a prefixpoint of $\left\lceil B\binom{\cdot\ \ y}{v}\right\rceil$. Because $\binom{x}{u}$ is the least such prefixpoint, it will then follow that $\binom{x}{u} \leq \binom{x'}{u'}$ and, in particular, $u \leq u'$. To construct this x', we consider the operator $\left\lceil B\binom{\cdot\ \ y}{u'\ \ v}\right\rceil$ on L_1 that maps every $z \in L_1$ to $\left\lceil B\binom{z\ \ y}{u'\ \ v}\right\rceil$. Because B is $\leq_p$-monotone, this is a monotone operator and, therefore, it must have a least fixpoint. Let x' be this least fixpoint. Because $u' \leq u$, $\binom{x\ \ y}{u'\ \ v} \leq_p \binom{x\ \ y}{u\ \ v}$ and $\left\lceil B\binom{x\ \ y}{u'\ \ v}\right\rceil \leq \left\lceil B\binom{x\ \ y}{u\ \ v}\right\rceil = x$, i.e., x is a prefixpoint of the operator $\left\lceil B\binom{\cdot\ \ y}{u'\ \ v}\right\rceil$. Therefore, $x' \leq x$. Now, because $\binom{x'\ \ y}{u'\ \ v} \leq_p \binom{x\ \ y}{u'\ \ v}$, $\left\lfloor B\binom{x'\ \ y}{u'\ \ v}\right\rfloor \leq \left\lfloor B\binom{x\ \ y}{u'\ \ v}\right\rfloor = B_1^{(x\ y)}(u') = u'$. Because, by construction, $x' = \left\lceil B\binom{x'\ \ y}{u'\ \ v}\right\rceil$, we have that $\left\lceil B\binom{x'\ \ y}{u'\ \ v}\right\rceil \leq \binom{x'}{u'}$ and $\binom{x'}{u'}$ is indeed a prefixpoint of $\left\lceil B\binom{\cdot\ \ y}{v}\right\rceil$. Therefore, $u \leq u'$. □

We now have all the material needed to prove that a stable fixpoint of A can be extended to a stable fixpoint of B.

Proof of Theorem 4.2. Let $(x\ y) = \mathcal{C}_A(x\ y)$ and let $(u\ v)$ be $\mathit{lfp}(B^{(x\ y)})$. We have to show that $\binom{x\ \ y}{u\ \ v}$ is a fixpoint of $\mathcal{C}_B$, i.e., that $\binom{x}{u} = C_B^{\uparrow}\binom{y}{v}$ and $\binom{y}{v} = C_B^{\downarrow}\binom{x}{u}$. We only prove the first equality; the proof of the other one is analogous. Let $\binom{x'}{u'}$ be $\mathit{lfp}(\left\lceil B\binom{\cdot\ \ y}{v}\right\rceil)$. We will show that $\binom{x'}{u'} = \binom{x}{u}$. By Theorem 4.1, we have that $\binom{x\ \ y}{u\ \ v}$ is a fixpoint of B, which implies that $\binom{x}{u}$ is a fixpoint of $\left\lceil B\binom{\cdot\ \ y}{v}\right\rceil$. Therefore,

$\binom{x'}{u'} \leq \binom{x}{u}$. We now show that also $\binom{x}{u} \leq \binom{x'}{u'}$. First, we prove that $x \leq x'$, by showing that x' is a prefixpoint of the operator $\lfloor A(\cdot\, y)|$, of which $x = C_A^{\downarrow}(y)$ is the least prefixpoint. If we let $(u''\ v'') = \mathit{lfp}(B^{(x'\ y)})$, then, because B is a fixpoint extension of A, $\lceil B(\begin{smallmatrix} x' & y \\ u'' & v'' \end{smallmatrix})\rceil = A(x'\ y)$. Moreover, because $\binom{x'}{u'}$ is a fixpoint of $\lfloor B(\begin{smallmatrix} \cdot & y \\ \cdot & v \end{smallmatrix})$, u' is a fixpoint of $B_1^{(x'\ y)}$. Since u'' is the least fixpoint of this operator, $u'' \leq u'$ and therefore $\binom{x'\ y}{u''\ v''} \leq_p \binom{x'\ y}{u'\ v''}$. By Proposition 4.3, the fact that $x' \leq x$ implies that $v'' = |\mathit{lfp}(B^{(x'\ y)})] \geq |\mathit{lfp}(B^{(x\ y)})] = v$. Consequently, we also have that $\binom{x'\ y}{u'\ v''} \leq_p \binom{x'\ y}{u'\ v}$ and, therefore, by $\leq_p$-monotonicity of B:

$$\lfloor A(x'\ y)| = \left\lceil B(\begin{smallmatrix} x' & y \\ u'' & v'' \end{smallmatrix})\right| \leq \left\lceil B(\begin{smallmatrix} x' & y \\ u' & v'' \end{smallmatrix})\right| \leq \left\lceil B(\begin{smallmatrix} x' & y \\ u' & v \end{smallmatrix})\right| = x'$$

Hence, x' is a prefixpoint of $\lfloor A(\cdot\ y)|$ and $x \leq x'$. Therefore, $x = x'$. Since $\binom{x'}{u'} = C_B^{\downarrow}(\binom{y}{v})$, Proposition 4.4 states that $u' = \mathit{lfp}(B_1^{(x'\ y)})$, which we now know to be identical to $\mathit{lfp}(B_1^{(x\ y)}) = u$. We conclude that $\binom{x}{u} = \binom{x'}{u'}$. □

So far, we have shown that for every stable fixpoint $(x\ y)$ of A, it must be the case that B has some stable fixpoint $\binom{x\ y}{u\ v}$. We are of course also interested in the converse question, i.e., in whether, for every stable fixpoint $\binom{x\ y}{u\ v}$ of B, the pair $(x\ y)$ is a stable fixpoint of A. In the beginning of this section, we already encountered an example showing that this is not always the case: transforming $\{P \leftarrow \neg\neg P\}$ into $\{P \leftarrow \neg N; N \leftarrow \neg P\}$ generates additional stable fixpoints, which do not correspond to fixpoints of the original rule set. To exclude such cases, we require B to be either part-to-whole or whole-to-part monotone.

Theorem 4.3. *Let B be a fixpoint extension of A that is either part-to-whole or whole-to-part monotone. If $\binom{x\ y}{u\ v}$ is a fixpoint of the stable operator $\mathcal{C}_B$, then $(x\ y)$ is a fixpoint of the stable operator $\mathcal{C}_A$ of A and $(u\ v) = \mathit{lfp}(B^{(x\ y)})$.*

Our proof of this result will make use of the following property of whole-to-part monotone operators.

Proposition 4.5. *Let B be an approximation on $L_1^2 \times L_2^2$. If B is whole-to-part monotone, then, for all $(x\ y) \in L_1^2$, the operator $B_1^{(x\ y)}$ coincides with $B_1^{(x\ \top)}$ and the operator $B_2^{(x\ y)}$ coincides with $B_2^{(\top\ y)}$.*

Proof. Let B and $(x\ y)$ be as above. To prove that $B_1^{(x\ y)} = B_1^{(x\ \top)}$, we observe that $\binom{x\ y}{u\ \top} \geq_p \binom{x\ \top}{u\ \top}$ and $\binom{x\ y}{u\ \top} \leq \binom{x\ \top}{u\ \top}$. It now follows from the $\leq_p$-monotonicity and whole-to-part monotonicity of B, that $\lfloor B(\begin{smallmatrix} x & y \\ u & \top \end{smallmatrix})| = \lfloor B(\begin{smallmatrix} x & \top \\ u & \top \end{smallmatrix})|$. The proof that $B_2^{(x\ y)} = B_2^{(\top\ y)}$ is analogous. □

We can now prove that part-to-whole or whole-to-part monotonicity indeed suffices to ensure that whenever $\binom{x\ y}{u\ v}$ is a stable fixpoint of B, then $(x\ y)$ is also a fixpoint of A.

Proof of Theorem 4.3. Let B be a fixpoint extension of A. We need to show that if $\binom{x\ y}{u\ v}$ is a fixpoint of $\mathcal{C}_B$, then $(x\ y)$ is a fixpoint of $\mathcal{C}_A$ and $(u\ v)$ is *lfp*$(B^{(x\ y)})$. By definition, $\binom{x\ y}{u\ v}$ is a fixpoint of $\mathcal{C}_B$ iff $\binom{x}{u} = C_B^{\downarrow}\binom{y}{v}$ and $\binom{y}{v} = C_B^{\uparrow}\binom{x}{u}$. By Proposition 4.4, if this is the case, then $u =$ *lfp*$(B_1^{(x\ y)})$ and $v =$ *lfp*$(B_2^{(x\ y)})$. Therefore, by Proposition 4.2, $(u\ v) =$ *lfp*$(B^{(x\ y)})$. What remains to be shown is that $x = C_A^{\downarrow}(y) =$ *lfp*$([A(\cdot\ y)|)$ and $y = C_A^{\uparrow}(x) =$ *lfp*$(|A(x\ \cdot)])$.

We will only prove the first equality; the proof of the second one is analogous. Because $\lfloor A(x\ y)| = \lceil B(\begin{smallmatrix}x & y\\ u & v\end{smallmatrix})| = x$, x is a fixpoint of $\lfloor A(\cdot\ y)|$. Let us now assume that there exists a fixpoint x' of this operator such that $x' \leq x$. We will show that $x \leq x'$, by constructing some u' for which $\binom{x'}{u'}$ is a prefixpoint of the operator $\left\lfloor B(\begin{smallmatrix}\cdot & y\\ & v\end{smallmatrix})\right|$, of which $\binom{x}{u}$ is the least fixpoint. Let $(u'\ v')$ be *lfp*$(B^{(x'\ y)})$. Observe that, by construction, we have that both $u' = \lfloor B(\begin{smallmatrix}x' & y\\ u' & v'\end{smallmatrix})$ and $x' = \lfloor A(x'\ y)| = \lceil B(\begin{smallmatrix}x' & y\\ u' & v'\end{smallmatrix})$, that is, $\binom{x'}{u'} = \left\lfloor B(\begin{smallmatrix}x' & y\\ u' & v'\end{smallmatrix})\right|$. We now need to distinguish between the case where B is part-to-whole monotone and the case where B is whole-to-part monotone.

- Suppose B is whole-to-part monotone. By Proposition 4.5, we have that the operators $B_2^{(x\ y)}$ and $B_2^{(x'\ y)}$ both coincide with $B_2^{(\top\ y)}$ and, therefore, they must have the same least fixpoint, i.e., $v = v'$. Therefore, $\binom{x'}{u'} = \left\lfloor B(\begin{smallmatrix}x' & y\\ u' & v'\end{smallmatrix})\right| = \left\lfloor B(\begin{smallmatrix}x' & y\\ u' & v\end{smallmatrix})\right|$. Because $\binom{x}{u}$ is the least fixpoint of $\left\lfloor B(\begin{smallmatrix}\cdot & y\\ & v\end{smallmatrix})\right|$, this implies that $\binom{x}{u} \leq \binom{x'}{u'}$ and, in particular, $x \leq x'$.

- Suppose B is part-to-whole monotone. Because $(x'\ y) \leq_p (x\ y)$, Proposition 4.3 implies that $(u'\ v') \leq_p (u\ v)$. In particular, $v' \geq v$. Because $(u'\ v') \geq (u'\ v)$, by part-to-whole monotonicity, $\binom{x'}{u'} = \left\lfloor B(\begin{smallmatrix}x' & y\\ u' & v'\end{smallmatrix})\right| \geq \left\lfloor B(\begin{smallmatrix}x' & y\\ u' & v\end{smallmatrix})\right|$. Therefore $\binom{x'}{u'}$ is a prefixpoint of $\left\lfloor B(\begin{smallmatrix}\cdot & y\\ & v\end{smallmatrix})\right|$ and, because $\binom{x}{u}$ is the least prefixpoint of this operator, $\binom{x}{u} \leq \binom{x'}{u'}$ and, in particular, $x \leq x'$.

□

Putting the above results together, we get the following theorem.

Theorem 4.4. *Let B be a fixpoint extension of A, such that B is either part-to-whole or whole-to-part monotone. $\binom{x\ y}{u\ v}$ is a stable fixpoint of B iff $(x\ y)$ is a stable fixpoint of A and $(u\ v) =$ lfp$(B^{(x\ y)})$. Moreover, $\binom{x\ y}{u\ v}$ is the well-founded fixpoint of B iff $(x\ y)$ is the well-founded fixpoint of A and $(u\ v) =$ lfp$(B^{(x\ y)})$.*

Proof. The correspondence between stable fixpoints follows from Theorems 4.2 and 4.3. By Proposition 4.3, this correspondence between stable fixpoints also implies the correspondence between well-founded fixpoints, as these are simply the $\leq_p$-least stable fixpoints. □

4.3 Application to logic programming

In this section, we use the algebraic results of Section 4.2 to derive a concrete equivalence theorem for rule sets. Recall that we are interested in transformations from some original rule set Δ over an alphabet Σ into a new rule set Δ' over an alphabet $\Sigma' = \Sigma \cup \sigma$. More concretely, Δ' is the result of replacing a subformula $\varphi(\mathbf{x})$ of some rule of Δ by a new predicate $P(\mathbf{x})$ and adding a new rule set δ to Δ to define this new predicate P. We will assume that $Def(\delta)$ contains no predicates that were already in Σ. Note that δ may also contain new open predicates, i.e., it is not necessarily the case that $Op(\delta) = \Sigma^p$. It is obvious that $Def(\Delta') = Def(\Delta) \cup Def(\delta)$. We will denote the result of replacing (some fixed set of occurrences of) $\varphi(\mathbf{x})$ in Δ by $P(\mathbf{x})$ as $\Delta[\varphi(\mathbf{x})/P(\mathbf{x})]$, i.e., $\Delta' = \Delta[\varphi(\mathbf{x})/P(\mathbf{x})] \cup \delta$.

We will now use our algebraic results to prove the equivalence of Δ' and Δ under a number of different semantics. Recall that these results relate the fixpoints of an approximation A on the square of some lattice L_1 to those of an operator B on the square of a product lattice $L_1 \times L_2$. In our case, we need to consider an pre-interpretation F and an interpretation O' extending F to the open predicates of Δ'. Our initial operator A will be the operator $\mathcal{T}^O_\Delta$, with $O = O'|_{Op(\Delta)}$, on the square of the lattice $\mathcal{L}^F_{Def(\Delta)}$ of interpretations for the defined predicates of Δ. The extended operator B will then be the operator $\mathcal{T}^{O'}_{\Delta'}$ on the square of the lattice $\mathcal{L}^F_{Def(\Delta')}$ of interpretations for $Def(\Delta')$. Because $Def(\Delta') = Def(\Delta) \cup Def(\delta)$, this last lattice is isomorphic to the product lattice $\mathcal{L}^F_{Def(\Delta)} \times \mathcal{L}^F_{Def(\delta)}$. Therefore, our lattices and operators are of the right form for our results to be applied.

Our first task is now to show that, under suitable conditions, $\mathcal{T}^{O'}_{\Delta'}$ is a fixpoint extension of $\mathcal{T}^O_\Delta$.

Theorem 4.5. *Let Δ be a rule set and let Δ' be the result $\Delta[\varphi(\mathbf{x})/P(\mathbf{x})] \cup \delta$ of replacing some $\varphi(\mathbf{x})$ in Δ by a new predicate $P(\mathbf{x})$ defined by some δ. Let O' be in $\mathcal{L}^F_{Op(\Delta')}$. Let $\mathcal{C}$ be the class of all structures that extend O' to Σ'. If the following conditions are satisfied:*

1. *δ is a monotone rule set and*
2. *for all $I, J \in \mathcal{C}$ such that $(I\ J) \models_s \delta$ it holds that for all $\mathbf{a} \in dom(F)$, $P(\mathbf{a})^{(I\ J)} = \varphi(\mathbf{a})^{(I\ J)}$,*

then $\mathcal{T}^{O'}_{\Delta'}$ is a fixpoint extension of $\mathcal{T}^O_\Delta$, with $O = O'|_{Op(\Delta)}$.

Proof. Let us first observe that, because the rules of Δ' with a new predicate in their head are precisely the rules of δ, we have that, for any $\left(\begin{smallmatrix} I_1 & J_1 \\ I_2 & J_2 \end{smallmatrix}\right) \in (\mathcal{L}^F_{Def(\Delta)} \times \mathcal{L}^F_{Def(\delta)})^2$:

$$\lfloor \mathcal{T}^{O'}_{\Delta'}\left(\begin{smallmatrix} I_1 & J_1 \\ I_2 & J_2 \end{smallmatrix}\right) \rfloor = (\mathcal{T}^{O'}_{\Delta'})^{(I_1\ J_1)}(I_2\ J_2) = \mathcal{T}^{(O' \cup I_1\ O' \cup J_1)}_\delta (I_2\ J_2). \tag{4.6}$$

Because δ is a monotone rule set, this shows that $\mathcal{T}^{O'}_{\Delta'}$ is part-to-part monotone. Now, let $(I_2\ J_2)$ be *lfp*$((\mathcal{T}^{O'}_{\Delta'})^{(I_1\ J_1)})$. We now need to show that

$$(\mathcal{T}^{O'}_{\Delta'})_{(I_2\ J_2)}(I_1\ J_1) = \mathcal{T}^O_\Delta(I_1\ J_1).$$

Because none of the rules in δ have a predicate from Σ in their head, these can safely be ignored, i.e., $(\mathcal{T}_{\Delta'}^{O'})_{(I_2\ J_2)}(I_1\ J_1) = (\mathcal{T}_{\Delta'\setminus\delta}^{O'})_{(I_2\ J_2)}(I_1\ J_1)$. Now, $\Delta' \setminus \delta$ and Δ are completely identical, apart from the fact that in some rule bodies of $\Delta' \setminus \delta$, the formula $\varphi(\mathbf{x})$ has been replaced by $P(\mathbf{x})$. Therefore, it now suffices to show that, for all tuples of domain elements $\mathbf{a} \in dom(F)^n$, $P(\mathbf{a})^{(I_2\ J_2)} = \varphi(\mathbf{a})^{(I_1\ J_1)}$. By equation (4.6), we clearly have that $(I_2\ J_2) = \mathit{lfp}(\mathcal{T}_{\delta}^{(I_1\ J_1)})$. Because δ is monotone, this implies that $\left(\begin{smallmatrix} I_1 & J_1 \\ I_2 & J_2 \end{smallmatrix}\right) \models_s \delta$. Condition 2 of the theorem now gives us precisely the equality we need. □

Note that, because we only consider monotone rule sets δ, we could have equivalently formulate the second condition of this theorem using the well-founded instead of the stable semantics, i.e., writing $\models_w$ instead of $\models_s$.

To see that this result indeed applies to Example 4.1 from our introduction (Section 4.1), let δ be the rule set given in (4.5), i.e.,

$$\delta = \left\{ \begin{array}{l} \forall r\ AllPrecHold(r) \leftarrow AllFrom(r,1). \\ \forall r,n\ AllFrom(r,n) \leftarrow \exists q\ Prec(r,n,q) \wedge Holds(q) \wedge AllFrom(r,n+1). \\ \forall r,n\ AllFrom(r,n) \leftarrow \exists q\ Prec(r,n,q) \wedge Holds(q) \wedge NbOfPrec(r,n). \end{array} \right\}.$$

Clearly, this δ is a positive rule set and, therefore, also monotone. Now, if we restrict our attention to those interpretations O' for $Op(\Delta')$ that actually correspond to $\mathcal{AL}$-rules[1], then it is easy to see that for all $r \in dom(O')$, $AllPrecHold(r)^{(I\ J)}$ iff $\varphi(r)^{(I\ J)}$. Hence, for such interpretations O', the second condition of Theorem 4.5 is also satisfied, and, therefore, the immediate consequence operator of the extended rule set is a fixpoint extension of the original operator.

Let us now look at some implications of Theorem 4.5. We first consider supported model and Kripke-Kleene semantics. By Theorem 4.1, we have that, if the definition δ is non-recursive, i.e., no new predicates appear in rule bodies, then the supported models and Kripke-Kleene model of Δ coincide with, respectively, the restriction of the supported models and Kripke-Kleene model of Δ' to the original alphabet Σ.

As we recall from Section 4.2, in order to also get a similar result for stable and well-founded semantics, we need some monotonicity properties. To this end, we will prove two results. The first result states that we get part-to-whole monotonicity if we replace only positively occurring subformulas, i.e., if after the transformation, the new predicates appear positively in the bodies of the original rules.

Theorem 4.6. *Let Δ be a rule set and let Δ' be the result $\Delta[\varphi(\mathbf{x})/P(\mathbf{x})] \cup \delta$ of replacing some $\varphi(\mathbf{x})$ in Δ by a new predicate $P(\mathbf{x})$ defined by some monotone rule set δ. Let O' be in $\mathcal{L}^F_{Op(\Delta')}$. If only positive occurrences of $\varphi(\mathbf{x})$ are replaced, then $\mathcal{T}_{\Delta'}^{O'}$ is part-to-whole monotone.*

Proof. Let $(I_1\ J_1)$ be in $(\mathcal{L}^F_{Def(\Delta)})^2$ and let $(I_2\ J_2)$ and $(I_2'\ J_2')$ be in $(\mathcal{L}^F_{Def(\delta)})^2$, such that $(I_2\ J_2) \leq (I_2'\ J_2')$. As can be seen from the proof of Theorem 4.5, the fact that δ is monotone implies that $\mathcal{T}_{\Delta'}^{O'}$ is part-to-part monotone. Therefore, it suffices to prove

[1]More specifically, for every r there should be a unique n such that $(r,n) \in NbOfPrec^{O'}$ and for every $1 \leq i \leq n$ there should be a unique q such that $(r,i,q) \in Prec^{O'}$.

that $\lceil \mathcal{T}_{\Delta'}^{O'}(\begin{smallmatrix} I_1 & J_1 \\ I_2 & J_2 \end{smallmatrix}) \rceil \leq \lceil \mathcal{T}_{\Delta'}^{O'}(\begin{smallmatrix} I_1 & J_1 \\ I_2' & J_2' \end{smallmatrix}) \rceil$. Because the new predicate $P(\mathbf{x})$ appears only positively in the rules that define the old predicates, this is the case.

□

For the purpose of eliminating universal quantifiers, we are only interested in replacing positively occurring subformulas, because a negatively occurring universal quantifier can of course simply be transformed into an existential one. Therefore, this kind of monotonicity will suffice for that particular purpose. However, in other applications, the following monotonicity result might also be useful. It states that we get whole-to-part monotonicity if the old predicates appear positively in the rules defining the new predicates.

Theorem 4.7. *Let Δ be a rule set and let Δ' be the result $\Delta[\varphi(\mathbf{x})/P(\mathbf{x})] \cup \delta$ of replacing some $\varphi(\mathbf{x})$ in Δ by a new predicate $P(\mathbf{x})$ defined by some δ. Let O' be in $\mathcal{L}^F_{Op(\Delta')}$. If the rules of δ contain only positive occurrences of atoms of $Def(\Delta)$, then $\mathcal{T}_{\Delta'}^{O'}$ is whole-to-part monotone.*

Proof. Because $Def(\Delta') = Def(\delta) \cup Def(\Delta)$ and δ is monotone, the condition of this theorem implies that δ is actually monotone in all of the predicates $Def(\Delta')$. Because the rules of δ are the only rules of Δ' with a new predicate in their head, this shows that $\mathcal{T}_{\Delta'}^{O'}$ is whole-to-part monotone. □

Put together, these two results tell us that the partial stable models and well-founded model of Δ coincide with the restrictions of, respectively, the partial stable models and well-founded model of Δ' to the original alphabet Σ, if the following conditions are satisfied. First, δ should be a monotone definition and all of its partial stable models should satisfy the four-valued equivalence between the new predicate $P(\mathbf{x})$ and the original formula $\varphi(\mathbf{x})$ (Condition 2 of Theorem 4.5). Second, it should either be the case that only positive occurrences of a formula are replaced, or that δ contains only positive occurrences of the original predicates.

Example 4.3. Let us consider, for instance, the following rule, representing an inertia property:

$$\forall p, t\ Holds(p, t+1) \leftarrow Holds(p, t) \wedge \neg(\exists a\ Occurs(a, t) \wedge Terminates(a, p)).$$

By our first monotonicity result (Theorem 4.6), we can replace the positively occurring formula $\neg\exists a\, Occurs(a, t) \wedge Terminates(a, p)$ by a new predicate $Unclipped(p, t)$, which gives us:

$$\forall p, t\ Holds(p, t+1) \leftarrow Holds(p, t) \wedge Unclipped(p, t).$$
$$\forall p, t\ Unclipped(p, t) \leftarrow \neg(\exists a\ Occurs(a, t) \wedge Terminates(a, p)).$$

By the second monotonicity result (Theorem 4.7), on the other hand, we can replace the formula $\exists a\ Occurs(a, t) \wedge Terminates(a, p)$ of the original inertia property—a negatively occurring formula that does not contain negation—by a new predicate $Clipped(a, p)$ in the following way:

$$\forall p, t\ Holds(p, t+1) \leftarrow Holds(p, t) \wedge \neg Clipped(p, t).$$
$$\forall p, t\ Clipped(p, t) \leftarrow \exists a\ Occurs(a, t) \wedge Terminates(a, p).$$

The results of this section show that both of these transformations preserve partial stable and well-founded models, and, since neither the rule for $Clipped$ nor $Unclipped$ is recursive, also that they preserve supported and Kripke-Kleene models.

As a final remark in this section, it is useful to come back to this four-valued equivalence between $P(\mathbf{x})$ as defined by δ and the original formula $\varphi(\mathbf{x})$, that is a condition of Theorem 4.5. One might wonder whether this is really necessary, i.e., whether it would suffice to check only the following two-valued equivalence:

$$\text{For all } I \in C \text{ such that } (I\ I) \models_s \delta : \forall \mathbf{a} \in D^n, I \models P(\mathbf{a}) \text{ iff } I \models \varphi(\mathbf{a}). \qquad (4.7)$$

In general, this is not the case. For instance, consider an attempt to replace in $\Delta = \{R \leftarrow Q \vee \neg Q;\ Q \leftarrow \neg Q\}$ the formula $\varphi = Q \vee \neg Q$ by a new predicate P, defined by a definition $\delta = \{P\}$. The above equivalence would then be satisfied and we would also get both part-to-whole and whole-to-part monotonicity, but there still is no correspondence between the models of Δ and Δ'. Indeed, the well-founded model of $\Delta' = \{R \leftarrow P;\ P;\ Q \leftarrow \neg Q\}$ is $(\{R, P\}\ \{R, P, Q\})$, while that of Δ is $(\{\}\ \{R, Q\})$.

The four-valued way of interpreting formulas is an integral part of both stable and well-founded semantics. Therefore, it makes sense that, as the above example shows, a four-valued equivalence is required in order to preserve either of these semantics. In practice, however, this should not pose too much of a problem, since most common transformations from classical logic, e.g. the De Morgan and distributivity laws, are still equivalence preserving in the four-valued case.

4.3.1 Applications and Related Work

The kind of transformations that we have considered in this chapter have a long history in logic programming. In this section, we will discuss some of this related work, while also pointing out a number of interesting applications of our results.

Predicate extraction and $\forall$-elimination

The following result is due to Van Gelder:

Theorem 4.8 (from (Van Gelder 1993)). *Let Δ be a rule set containing a rule $r = \forall \mathbf{x}\ P(\mathbf{t}) \leftarrow \psi$. Let $\varphi(\mathbf{y})$ be an existentially quantified conjunction of literals, and let Q be a new predicate symbol. If $\varphi(\mathbf{y})$ is a positively occurring subformula of ψ, then Δ is equivalent under the partial stable and well-founded semantics to the rule set Δ', that results from replacing $\varphi(\mathbf{y})$ in r by $Q(\mathbf{y})$ and adding the rule $\forall \mathbf{y}\ Q(\mathbf{y}) \leftarrow \varphi(\mathbf{y})$ to Δ.*

Because the rule set $\delta = \{\forall \mathbf{y}\ Q(\mathbf{y}) \leftarrow \varphi(\mathbf{y})\}$ clearly satisfies the conditions of Theorem 4.5, Van Gelder's theorem follows directly from ours. This result provides a theoretical justification for the common programming practice of *predicate extraction*: replacing a subformula that occurs in multiple rules by a new predicate to make the program more concise and more readable. In (Schrijvers and Serebrenik 2004), predicate extraction is considered to be an important *refactoring* operation (i.e., an equivalence preserving transformation to improve maintainability) for logic programming.

Our result extends Van Gelder's theorem by allowing the new predicate Q to be defined by an additional rule set δ, instead of allowing only the definition $\{\forall \mathbf{y}\ Q(\mathbf{y}) \leftarrow \varphi(\mathbf{y})\}$. In particular, *recursive* definitions of Q are also allowed. This significantly increases the applicability of the theorem. Indeed, as we already illustrated in the introduction, this allows us to eliminate universal quantifiers. The general idea behind this method is that we can replace a universal quantifier by a recursion over some total order on the domain. Of course, this can only be done if the domain in question is finite.

Definition 4.3 (Domain iterator). Let C be a set of Σ-structures with domain D. Let $First/1$, $Next/2$ and $Last/1$ be predicate symbols of Σ. We will call the triple $\langle First, Next, Last\rangle$ a *domain iterator* in C iff for each structure $S \in C$: the transitive closure of $Next^S$ is a total order on D and there exists elements $f, l \in D$ such that $First^S = \{f\}$ and $Last^S = \{l\}$ and, for all $x \in D \setminus \{l\}$ there exists a unique $y \in D \setminus \{f\}$ such that $(x, y) \in Next^S$.

Given such a domain iterator $It = \langle First, Next, Last\rangle$, we can introduce the following rule set δ_φ^{It} to define a new predicate $Forall(\mathbf{x})$ as a replacement for some positive occurrence of a formula $\varphi(\mathbf{x}) = \forall y\ \psi(\mathbf{x}, y)$:

$$\begin{aligned} \forall \mathbf{x}, y\ Forall(\mathbf{x}) &\leftarrow First(y) \wedge \psi(\mathbf{x}, y) \wedge AllFrom(\mathbf{x}, y).\\ \forall \mathbf{x}, y\ AllFrom(\mathbf{x}, y) &\leftarrow Next(y, y') \wedge \psi(\mathbf{x}, y') \wedge AllFrom(\mathbf{x}, y').\\ \forall \mathbf{x}, y\ AllFrom(\mathbf{x}, y) &\leftarrow Last(y). \end{aligned} \tag{4.8}$$

It is quite obvious that, for non-empty, finite domains, this transformation satisfies the conditions of Theorems 4.5 and 4.6. Therefore, we directly get the following result.

Theorem 4.9 ($\forall$ elimination). *Let Δ be a rule set and $\varphi(\mathbf{x})$ be a formula of the form $\forall y\ \psi(\mathbf{x}, y)$, that appears only positively in the bodies of rules of Δ. For a set of structures C with finite domain, if It is a domain iterator, then $\Delta[\varphi/Forall] \cup \delta_\varphi^{It}$ is equivalent to Δ under partial stable and well-founded semantics.*

This theorem provides a way of eliminating universal quantifiers from rule bodies under the stable and well-founded semantics. As we already pointed out in Section 4.1, this offers an alternative to the corresponding step from the Lloyd-Topor transformation (Lloyd and Topor 1984), which is valid only for the two-valued completion semantics and not for stable or well-founded semantics.

In the above theorem, we assume a total order on the entire domain and this same order can be used to eliminate all universally quantified formulas, that satisfy the condition of the theorem. This is not precisely what happened in our motivating example. Indeed, there, the universally quantified formula $\varphi(\mathbf{x})$ was of the form: $\forall \mathbf{y}\ \Psi_1(\mathbf{x}, \mathbf{y}) \Rightarrow \Psi_2(\mathbf{x}, \mathbf{y})$. Using the above theorem, we would replace $\varphi(\mathbf{x})$ by a recursion that says that the implication $\Psi_1(\mathbf{x}) \Rightarrow \Psi_2(\mathbf{x})$ must hold for every element in the domain. However, in our original version of this example, we actually replaced $\varphi(\mathbf{x})$ by a recursion which says that for all $\mathbf{y}$ that satisfy $\Psi_1(\mathbf{x}, \mathbf{y})$ (i.e., for all i, q such that $Prec(r, i, q)$) the consequent $\Psi_2(\mathbf{x}, \mathbf{y})$ (i.e., $Holds(q)$) is satisfied. This is a more fine-grained approach, which we can also prove in general.

A *restricted iterator* for $\mathbf{y}$ of $\psi_1(\mathbf{x},\mathbf{y})$ in a structure I is a triple of predicates $\langle First(\mathbf{x},\mathbf{y}), Next(\mathbf{x},\mathbf{y},\mathbf{y}'), Last(\mathbf{x},\mathbf{y})\rangle$, such that for all tuples $\mathbf{d}$ of elements of the domain D of I, $\langle First(\mathbf{d},\mathbf{y}), Next(\mathbf{d},\mathbf{y},\mathbf{y}'), Last(\mathbf{d},\mathbf{y})\rangle$ is an iterator over $\{\mathbf{e} \in D^n \mid I \models \Psi_1(\mathbf{d},\mathbf{e})\}$. Given such a restricted iterator, we can define the following replacement $Forall(\mathbf{x})$ for $\varphi(\mathbf{x})$:

$$\begin{aligned}
\forall \mathbf{x},\mathbf{y}\ Forall(\mathbf{x}) &\leftarrow First(\mathbf{x},\mathbf{y}) \wedge \Psi_2(\mathbf{x},\mathbf{y}) \wedge AllFrom(\mathbf{x},\mathbf{y}).\\
\forall \mathbf{x},\mathbf{y},\mathbf{y}'\ AllFrom(\mathbf{x},\mathbf{y}) &\leftarrow Next(\mathbf{x},\mathbf{y},\mathbf{y}') \wedge \Psi_2(\mathbf{x},\mathbf{y}') \wedge AllFrom(\mathbf{x},\mathbf{y}').\\
\forall \mathbf{x},\mathbf{y}\ AllFrom(\mathbf{x},\mathbf{y}) &\leftarrow Last(\mathbf{x},\mathbf{y}).
\end{aligned}$$

Again, Theorem 4.5 can be used to show that, if there is at least one tuple that satisfies Ψ_1, $\varphi(\mathbf{x})$ can be replaced by $Forall(\mathbf{x})$.

The idea behind such a restricted iterator is very similar to that of a *bounded universal quantifier*, i.e., one which quantifies only over a subset of the domain. There have been a number of publications studying bounded quantifiers in the context of logic programs. For instance, (Voronkov 1992) examines how such quantifiers can be added to definite logic programs. The difference with our results presented above is that we do not consider the addition of a new language construct, but instead show how the bounded quantification effect can be achieved using a simple recursive definition of a new predicate.

Above, we discussed a result by Van Gelder, shown here as Theorem 4.8, about predicate extraction for positively appearing subformula. In (Van Gelder 1993), Van Gelder also considered predicate extraction for *negatively* occurring subformulas. His results on this topic are, however, substantially different from our Theorem 4.7. Indeed, we prove that, if Δ' is the result of performing predicate introduction on a rule set Δ, then, under certain conditions, the restriction of the well-founded model of Δ' to the original alphabet coincides with the well-founded model of Δ. Van Gelder's results, on the other hand, prove that in *all* cases certain *parts* of the well-founded model of Δ and Δ' will be the same. This result is not implied by ours, nor vice versa. We have actually found no results similar to our Theorem 4.7 in the literature.

Fold/unfold transformations and partial evaluation

There is a long tradition in logic programming of studying transformations that preserve the semantics of a program (or parts of it), but make, for instance, its execution more efficient. We refer to (Pettorossi and Proietti 1994) for an overview. Fold/unfold transformations play an important role in this context. A lot of this work was originally carried out for definite programs, but has later been extended to normal programs as well. For instance, (Aravindan and Dung 1995) proves the correctness of fold/unfold transformations for stable and well-founded semantics.

In the literature, there are a number of variants of folding, depending on which predicates are allowed to be folded. In (Gardner and Shepherdson 1991; Maher 1993), folding is defined approximately as follows. It is a transformation that takes two rules:

$$\forall \mathbf{x}\ H \leftarrow Body_1 \wedge Body_2. \tag{4.9}$$

$$\forall \mathbf{x}\ A \leftarrow Body'. \tag{4.10}$$

with $Body_1, Body_2$ and $Body'$ conjunctions of literals and A, H atoms, such that:

- There exists a substitution θ such that $Body_1 = Body'\theta$;
- For this θ, $A\theta$ unifies with no other head of a rule of the program.

It then transforms these into:

$$\forall \mathbf{x}\ H \leftarrow A\theta \wedge Body_2.$$
$$\forall \mathbf{x}\ A \leftarrow Body'.$$

Let us assume also that $Body_1$ does not contain an atom that unifies with A and let $A = P(\mathbf{t})$. The fact that $A\theta$ unifies with no other head of the program means that it essentially acts as though it were an atom containing a new predicate. More precisely put, if we were to perform a predicate introduction step using some new predicate Q and replace $Body_1$ in (4.9) by $Q(\mathbf{t}\theta)$ defined by $\delta = \{(\forall \mathbf{x}\ Q(\mathbf{t}) \leftarrow Body')\theta\}$, then it is easy to see that the result of this will be equivalent to the result of the folding step. If, however, $Body_1$ itself already contains an atom that unifies with $A\theta$, then it is no longer possible to view this as a new atom and, consequently, the correspondence to predicate introduction will be lost. In (Tamaki and Sato 1984; Seki 1993), a slightly different kind of folding is considered, which cannot completely be seen as a solitary transformation from one program into another; instead, folding is defined in the context of a longer sequence of transitions between programs. As such, there is no direct correspondence to predicate introduction in this case.

Unfolding is, roughly speaking, the inverse of folding. To be more precise, an unfold transformation takes a normal logic programming rule of the form

$$\forall \mathbf{x}\ H \leftarrow A \wedge Body. \tag{4.11}$$

where H and A are atoms and $Body$ is a conjunction of literals, and transforms this as follows. Let the following be all rules of the program whose head unifies with H:

$$\forall \mathbf{x}\ A_1 \leftarrow Body_1.$$
$$\ldots$$
$$\forall \mathbf{x}\ A_n \leftarrow Body_n.$$

For each i, let θ_i be the most general unifier of A and A_i. The rule (4.11) is then transformed into

$$(\forall \mathbf{x}\ H \leftarrow Body_1 \wedge Body)\theta_1$$
$$\ldots$$
$$(\forall \mathbf{x}\ H \leftarrow Body_n \wedge Body)\theta_n$$

For the relation between unfolding and predicate introduction, we note in a similar way that if an unfolding step is performed on a predicate that appears nowhere in $Body$

or one of the $Body_i$'s, then this again acts as though it were a new predicate and the unfolding transformation corresponds to the inverse of a predicate introduction step. In (Dix 1995), the *principle of partial evaluation* was introduced as a means of comparing different semantics for logic programs. Essentially, a semantics satisfies this principle if it is equivalence preserving for this restricted class of unfolding transformations. Therefore, our theorem shows that the stable and well-founded model semantics both satisfy this principle, which was first proven in (Dix 1995).

Our results go beyond this work on folding/unfolding in two main ways. First, we have studied rule sets, that may contain arbitrary first-order formulas in the bodies and may have open predicates. Second, and more importantly, fold/unfold transformations only manipulate formulas that are already present in the program, whereas we allow definitions for a new predicate to, for instance, contain recursion where originally there was none. Both these aspects are crucial for our method of eliminating of universal quantifiers.

A related class of program transformation techniques are those of *partial evaluation* (Gallagher 1993). Essentially, an unfolding step is already a particular form of partial evaluation, and even a very important one. In general, however, partial evaluation is more than just unfolding, since it also considers transformations that can take into account the fact that a certain part of the "input" has already been fixed, i.e., they are only interested in preserving a particular *part* of the semantics of a rule set. This is a kind of equivalence that we have not considered.

More recently, there has been a lot of work in Answer Set programming on the topic of strong equivalence. Two rule sets Δ and Δ' are called strongly equivalent iff for all rule sets Δ'', it is the case that $\Delta \cup \Delta''$ and $\Delta' \cup \Delta''$ are equivalent, in the normal sense of having the same stable models. Technically speaking, the transformations we consider here do not even preserve normal equivalence, due to the fact that they introduce new predicates. Indeed, our results only concern equivalence w.r.t. the original alphabet Σ of a rule set. Even this equivalence, however, can be lost if we allow the introduction of additional rules. For instance, our result shows that $\Delta = \{P \leftarrow \neg Q; Q \leftarrow Q\}$ is equivalent to $\Delta' = \{P \leftarrow R; R \leftarrow \neg Q; Q \leftarrow Q\}$ w.r.t. the original alphabet $\{P, Q\}$. However, if we now add to both Δ and Δ' the rule set $\Delta'' = \{Q \leftarrow \neg R; R \leftarrow R\}$, we see that the restrictions to Σ of the stable models of $\Delta \cup \Delta''$ and $\Delta' \cup \Delta''$ are not the same. Indeed, $\Delta' \cup \Delta''$ has a stable model in which P holds, whereas $\Delta \cup \Delta''$ does not.

4.4 Application to autoepistemic logic

In this section, we use our algebraic results to study the problem of predicate introduction in autoepistemic logic. Concretely, the goal of the transformations we consider is to eliminate nested modal operators by the introduction of a new propositional symbol. We begin with an informal analysis of such transformations.

4.4.1 Introduction to the problem

As an example, let us consider the following formula:

$$t \Rightarrow K(Kr \Rightarrow Ks).$$

This formula states that, under some condition t, the reasoner knows that knowledge about r implies knowledge about s.

Now, suppose we introduce some proposition p, not belonging to the original alphabet $\Sigma = \{t, r, s\}$ of this formula and that we would like p to mean "s is known". In general, we have a formula F of a theory T and want to replace a subformula $K\varphi$ (here, $\varphi = s$), that appears inside the scope of some other modal operator. We can assume without loss of generality that $T = \{F\}$, because every theory is equivalent to the singleton theory consisting of the conjunction of its formulas. Let ψ be the smallest modal literal of F that contains $K\varphi$. In our example, $\psi = K(Kr \Rightarrow Ks)$. Let F' be the result of replacing $K\varphi$ by p, i.e., in this case $F' = (t \Rightarrow K(Kr \Rightarrow p))$. Intuitively, what we want to do now is construct a formula F_p that defines the new atom p in such a way that the models (under some semantics) of the original theory T coincide with the restrictions to the original alphabet Σ of the models of the new theory $T' = \{F', F_p\}$. We will now give some intuitions on how we should construct such an F_p for the above example.

Perhaps the most obvious candidate formula would be $Ks \Leftrightarrow p$, which abbreviates $(Ks \Rightarrow p) \wedge (Ks \Leftarrow p)$. However, it turns out that this will not work. If we abbreviate the set of all interpretations $\mathcal{I}_{\{t,r,s,p\}}$ for the alphabet of T' as $\mathcal{I}$, we see that T' has a partial expansion $(\mathcal{I}\ \{\})$. Indeed, on the one hand, $T'\langle\{\}\ \mathcal{I}\rangle = \{t \Rightarrow \mathbf{t}; \mathbf{f} \Rightarrow p; \mathbf{t} \Leftarrow p\}$, whose set of models is $\mathcal{I}$, while, on the other hand, $T'\langle\mathcal{I}\ \{\}\rangle$ contains both the formula $\mathbf{t} \Rightarrow p$ and $\mathbf{f} \Leftarrow p$ and, as such, has no models. However, the corresponding pair $(\mathcal{I}_{\{r,t,s\}}\ \{\})$ is not a partial expansion of the original formula F, because $F\langle\mathcal{I}_{\{r,t,s\}}\ \{\}\rangle = (t \Rightarrow \mathbf{f})$, of which every interpretation in which t is false is a model, so $Mod(t \Rightarrow \mathbf{f}) \neq \{\}$. We therefore need to look for a different formula F_p.

It is clear that, in order to ensure that we do get the result we want, it would suffice to get $\psi = K(Kr \Rightarrow Ks)$ to be equivalent to $\psi' = K(Kr \Rightarrow p)$. Now, ψ holds iff it is the case that either there is a possible world in which r is false or in all possible worlds, s is true. The formula ψ', on the other hand, holds iff it is the case that either there is a possible world in which r is false or in all possible worlds, p is true. As such, the formula F_p should force p to be known iff originally s was known. This suggests the formula $Ks \Rightarrow p$. Indeed, if it was originally the case that Ks, then the only possible world for p will be $\{p\}$, but otherwise both $\{\}$ and $\{p\}$ will be possible and p will not be known. A similar line of reasoning applies whenever we want to replace a formula $K\varphi$ that appears *positively* within the scope of the smallest modal literal ψ that contains it.

To illustrate the other case, let us now try to replace Kr in the above formula F by a new atom q. Once again, it suffices to ensure that ψ is equivalent to the formula $\psi'' = K(q \Rightarrow Ks)$. Now, ψ'' holds iff either in all possible worlds q is false or in all possible worlds s is true. As such, our formula F_q should, in this case, make sure that q is known to be false whenever, originally, r was not known to be true. That is to say, if there exists a world in which r is false, then q should be false in all worlds. This suggests the formula $\neg Kr \Rightarrow \neg q$, which is of course simply a rewriting of $Kr \Leftarrow q$.

Once again, a similar line of reasoning applies whenever we are trying to replace a formula $K\varphi$ that appears *negatively* within the scope of the smallest modal literal ψ that contains it.

Let us now formally define the problem that we want to consider.

Definition 4.4. Let $T = \{F\}$ be an autoepistemic theory. We consider an occurrence of a modal literal $K\varphi$ with K-rank at least 1. Let p be a proposition that does not belong to the alphabet of T. The result of *introducing p to replace (this occurrence of) $K\varphi$* is the theory $T' = \{F', F_p\}$, where F' is the result of replacing the selected occurrence of $K\varphi$ in F by p and F_p is defined as follows, depending on how $K\varphi$ appears inside the scope of the smallest modal literal ψ that contains it:

- If $K\varphi \in^+ \psi$, then $F_p = (K\varphi \Rightarrow p)$;
- If $K\varphi \in^- \psi$, then $F_p = (K\varphi \Leftarrow p)$.

The question we we will study is when (and for which of the previously mentioned semantics) the result of introducing p to replace $K\varphi$ will be equivalent to the original theory. The rest of this section will continue to use the notations introduced in the above definition. We will also use Σ to denote the alphabet of the original theory T and Σ' to denote the alphabet $\Sigma \cup \{p\}$ of T'. By ψ' we will denote the result of replacing $K\varphi$ by p in ψ.

4.4.2 Application of the algebraic results

We now apply our algebraic theorems to the problem at hand. Recall that these relate an approximation A on a lattice L_1^2 to a fixpoint extension B of A, which is an operator on a lattice $L_1^2 \times L_2^2$. In the current case, L_1^2 will be the lattice $\mathcal{B}_\Sigma$ of pairs of possible worlds structures for the original alphabet Σ and L_2^2 will be the lattice $\mathcal{B}_{\{p\}}$ of pairs of possible world structures for the new alphabet $\{p\}$. Therefore, to play the role of the fixpoint extension B, we need an operator on the square $\tilde{\mathcal{B}}_{\Sigma'}$ of the product lattice $\tilde{\mathcal{W}}_{\Sigma'} = \mathcal{W}_\Sigma \times \mathcal{W}_{\{p\}}$. Of course, this gives us the same problem as we encountered in Section 3.4, namely that this lattice is not isomorphic to the lattice $\mathcal{B}_{\Sigma'}$ on which $\mathcal{D}_{T'}$ operates. Here, we will solve this problem in the same way, namely, by also defining an intermediate operator $\tilde{\mathcal{D}}_{T'}$ on $\tilde{\mathcal{B}}_{\Sigma'}$.

Let us first recall that, in Section 3.4, we defined a function κ from $\tilde{\mathcal{W}}_{\Sigma'}$ to $\mathcal{W}_{\Sigma'}$ that maps each pair $\binom{X}{U}$ to $\{I \cup J \mid I \in X \text{ and } J \in U\}$. We first observe that, just as before, none of the possible world sets outside of $\kappa(\tilde{\mathcal{W}}_{\Sigma'})$ are relevant for the operator $\mathcal{D}_{T'}$. Indeed, for any belief pair $(P'\ S')$ in the image of this operator, both P' and S' will be of the form $Mod(T'\langle P\ S\rangle)$ for some belief pair $(P\ S)$. Now, every $T'\langle P\ S\rangle$ is a—classical—propositional theory, which consists of a formula $F'\langle P\ S\rangle$ in alphabet Σ, and a formula $F_p\langle P\ S\rangle$ in alphabet $\{p\}$. Because these two alphabets are disjoint, it is an obvious property of propositional logic that $Mod(T'\langle P\ S\rangle)$ consists of all $I \cup J$ for which I is a model of $F'\langle P\ S\rangle$ and J is a model of $F_p\langle P\ S\rangle$. In other words, for any $(P\ S)$, $Mod(T'\langle P\ S\rangle) = \kappa(\binom{Mod(F'\langle P\ S\rangle)}{Mod(F_p\langle P\ S\rangle)})$. We therefore conclude that $\mathcal{D}_{T'}(\tilde{\mathcal{B}}_{\Sigma'}) \subseteq \overline{\kappa}(\tilde{\mathcal{B}}_{\Sigma'})$.

We will now summarize some relevant properties of κ. Recall that in Section 3.4, we introduced the notation $\tilde{\mathcal{W}}^c_{\Sigma'}$ for the set of all consistent elements of $\tilde{\mathcal{W}}_{\Sigma'}$, that is, all $\binom{X}{U}$ for which both $X \neq \{\}$ and $U \neq \{\}$ or, equivalently, $\kappa(\binom{X}{U}) \in \mathcal{W}^c_{\Sigma'}$. By $\tilde{\mathcal{W}}^{\downarrow c}_{\Sigma'}$ we denote the set of all $\binom{X}{U}$ for which $U \neq \{\}$ and $\tilde{\mathcal{W}}^{\uparrow c}_{\Sigma'}$ denotes the set of all $\binom{X}{U}$ for which $X \neq \{\}$. We use similar notations $\tilde{\mathcal{B}}^c_{\Sigma'}$, $\tilde{\mathcal{B}}^{\downarrow c}_{\Sigma'}$ and $\tilde{\mathcal{B}}^{\uparrow c}_{\Sigma'}$ for the squares of these lattices.

Proposition 4.6. *The function κ has the following properties:*

1. *κ is injective on the subset $\tilde{\mathcal{W}}^c_{\Sigma'}$ of its domain;*
2. *For all $\binom{X}{U} \in \tilde{\mathcal{W}}^{\downarrow c}_{\Sigma'}$, $\kappa(\binom{X}{U})|_{\Sigma} = X$ and for all $\binom{X}{U} \in \tilde{\mathcal{W}}^{\uparrow c}_{\Sigma'}$, $\kappa(\binom{X}{U})|_{\{p\}} = U$;*
3. *$\mathcal{D}_{T'}(\tilde{\mathcal{B}}_{\Sigma'}) \subseteq \overline{\kappa}(\tilde{\mathcal{B}}_{\Sigma'})$.*

Let us now define an intermediate operator $\tilde{\mathcal{D}}_{T'}$ on $\tilde{\mathcal{B}}_{\Sigma'}$, such that, on the one hand, this $\tilde{\mathcal{D}}_{T'}$ is a fixpoint extension of the operator $\mathcal{D}_T$ and, on the other hand, the fixpoints and stable fixpoints of $\tilde{\mathcal{D}}_{T'}$ correspond to those of $\mathcal{D}_{T'}$.

Definition 4.5. We define the function $\tilde{\mathcal{D}}^u_T$ from $\tilde{\mathcal{B}}_{\Sigma'}$ to $\tilde{\mathcal{W}}_{\Sigma'}$ as mapping every $\binom{X\ Y}{U\ V}$ to the pair $\binom{X'}{U'}$ for which:

- $X' = Mod(F'\langle\overline{\kappa}(\binom{X\ Y}{U\ V})\rangle)$;
- $U' = Mod(F_p\langle X\ Y\rangle)$

We also define $\tilde{\mathcal{D}}_T(\binom{X\ Y}{U\ V})$ as $(\tilde{\mathcal{D}}^u_T(\binom{Y\ X}{V\ U})\ \tilde{\mathcal{D}}^u_T(\binom{X\ Y}{U\ V}))$ and $\tilde{D}_T(\binom{X}{U})$ as $\tilde{\mathcal{D}}^u_{T'}(\binom{X\ X}{U\ U})$.

This operator $\tilde{\mathcal{D}}_{T'}$ differs from $\mathcal{D}_{T'}$ in two respects. First, in the construction of a new belief pair for alphabet Σ, it considers only the formula F', whereas, in the construction of a new belief pair for $\{p\}$, it only considers F_p. Second, a new belief pair for $\{p\}$ is constructed using only the original belief pair $(X\ Y)$ for Σ, instead of the entire belief pair $\overline{\kappa}(\binom{X\ Y}{U\ V})$. Because of this, every operator $\tilde{\mathcal{D}}^{(X\ Y)}_{T'} = \lfloor\tilde{\mathcal{D}}_{T'}(\binom{X\ Y}{\cdot\ \cdot})\rfloor$ is actually constant. We now investigate how we can characterize the pair $(U\ V)$ that is the unique element in the image of some $\tilde{\mathcal{D}}^{(X\ Y)}_{T'}$.

First, let us observe that if $K\varphi \in^+ \psi$, we can determine $Mod(F_p\langle X\ Y\rangle)$ as follows:

$K\varphi \in^+ \psi$	$K\varphi\langle Y\ X\rangle = \mathbf{t}$	$K\varphi\langle Y\ X\rangle = \mathbf{f}$
F_p	$K\varphi \Rightarrow p$	$K\varphi \Rightarrow p$
$F_p\langle X\ Y\rangle$	$\mathbf{t} \Rightarrow p$	$\mathbf{f} \Rightarrow p$
$Mod(F_p\langle X\ Y\rangle$	$\{\{p\}\}$	$\{\{\}, \{p\}\}$

For $K\varphi \in^- \psi$, the analogous table is as follows:

$K\varphi \in^- \psi$	$K\varphi\langle X\ Y\rangle = \mathbf{t}$	$K\varphi\langle X\ Y\rangle = \mathbf{f}$
F_p	$K\varphi \Leftarrow p$	$K\varphi \Leftarrow p$
$F_p\langle X\ Y\rangle$	$\mathbf{t} \Leftarrow p$	$\mathbf{f} \Leftarrow p$
$Mod(F_p\langle X\ Y\rangle$	$\{\{\}, \{p\}\}$	$\{\{\}\}$

By definition, if $(U\ V)$ is the unique element in the image of some $\tilde{\mathcal{D}}_{T'}^{(X\ Y)}$, then U is $Mod(F_p\langle Y\ X\rangle)$ and V is $Mod(F_p\langle X\ Y\rangle)$, so we can find the precise values of these two possible world structures by means of the above tables. We remark that, in particular, it is always the case that both $U \neq \{\}$ and $V \neq \{\}$, i.e., $\tilde{\mathcal{D}}_{T'}(\tilde{\mathcal{B}}_{\Sigma'}) \subseteq \tilde{\mathcal{B}}_{\Sigma'}^{\downarrow c}$.

We are of course mainly interested in fixpoints of $\tilde{\mathcal{D}}_{T'}$. Clearly, for every such fixpoint $\binom{X\ Y}{U\ V}$, it has to be the case that $(U\ V)$ is the unique element in the image of $\tilde{\mathcal{D}}_{T'}^{(X\ Y)}$. Let us denote by $\tilde{\mathcal{B}}_{\Sigma'}^{*}$ the set of all $\binom{X\ Y}{U\ V}$ for which this last property holds. It can easily be seen from the above discussion that, in every such element of $\tilde{\mathcal{B}}_{\Sigma'}^{*}$, the values of U and V are as follows.

Proposition 4.7. *For all $\binom{X\ Y}{U\ V} \in \tilde{\mathcal{B}}_{\Sigma'}^{*}$, the values of U and V depend on the values of X and Y, and on how $K\varphi$ appears in ψ, as given by the following table:*

	$K\varphi\langle X\ Y\rangle$		$K\varphi\langle Y\ X\rangle$	
	t	**f**	**t**	**f**
$K\varphi \in^{+} \psi$	$U = \{\{p\}\}$	$U = \{\{\}, \{p\}\}$	$V = \{\{p\}\}$	$V = \{\{\}, \{p\}\}$
$K\varphi \in^{-} \psi$	$V = \{\{\}, \{p\}\}$	$V = \{\{\}\}$	$U = \{\{\}, \{p\}\}$	$U = \{\{\}\}$

Relating fixpoints of $\tilde{\mathcal{D}}_{T'}$ and $\mathcal{D}_{T'}$

We now show that $\tilde{\mathcal{D}}_{T'}$ and $\mathcal{D}_{T'}$ have the same fixpoints. Because every fixpoint of $\tilde{\mathcal{D}}_{T'}$ must obviously belong to its image, it suffices to consider only $\tilde{\mathcal{B}}_{\Sigma'}^{\downarrow c}$, i.e., the set of all $\binom{X\ Y}{U\ V} \in \tilde{\mathcal{B}}_{\Sigma'}$ for which $U, V \neq \{\}$. On this particular part of its domain, the operator $\tilde{\mathcal{D}}_{T'}$ is related to $\mathcal{D}_{T'}$ in the following way.

Proposition 4.8. *For all $\binom{X\ Y}{U\ V} \in \tilde{\mathcal{B}}_{\Sigma'}^{\downarrow c}$, $\overline{\kappa}(\tilde{\mathcal{D}}_{T'}(\binom{X\ Y}{U\ V})) = \mathcal{D}_{T'}(\overline{\kappa}(\binom{X\ Y}{U\ V}))$.*

Proof. Let $\binom{X\ Y}{U\ V} \in \tilde{\mathcal{B}}_{\Sigma'}^{\downarrow c}$ and let $(P\ S)$ be $\overline{\kappa}(\binom{X\ Y}{U\ V})$. As we already showed in the discussion leading up to Proposition 4.6, $Mod(T'\langle P\ S\rangle) = \kappa(\binom{Mod(F'\langle P\ S\rangle)}{Mod(F_p\langle P\ S\rangle)})$. Therefore, it suffices to show that $F_p\langle P\ S\rangle = F_p\langle X\ Y\rangle$. Because $U, V \neq \{\}$, we have that $(P\ S)|_{\Sigma} = (X\ Y)$, which proves the result. □

From this proposition, the correspondence between fixpoints now follows.

Theorem 4.10. *For all $(P\ S) \in \mathcal{B}_{\Sigma}$, $(P\ S)$ is a fixpoint (respectively, the Kripke-Kleene fixpoint) of $\mathcal{D}_{T'}$ iff there exists a $\binom{X\ Y}{U\ V} \in \tilde{\mathcal{B}}_{\Sigma'}$ such that $\overline{\kappa}(\binom{X\ Y}{U\ V}) = (P\ S)$ and $\binom{X\ Y}{U\ V}$ is a fixpoint (the Kripke-Kleene fixpoint) of $\tilde{\mathcal{D}}_{T'}$.*

Proof. Every fixpoint $\binom{X\ Y}{U\ V}$ of $\tilde{\mathcal{D}}_{T'}$ must belong to $\tilde{\mathcal{B}}_{\Sigma'}^{\downarrow c}$ and therefore it follows directly from Proposition 4.8 that $\overline{\kappa}(\binom{X\ Y}{U\ V})$ is a fixpoint of $\mathcal{D}_{T'}$. To prove the other direction, we must show that for every fixpoint $(P\ S)$ of $\mathcal{D}_{T'}$, there exists a fixpoint $\binom{X\ Y}{U\ V}$ of $\tilde{\mathcal{D}}_{T'}$ such that $\overline{\kappa}(\binom{X\ Y}{U\ V}) = (P\ S)$. Let $(X\ Y)$ be $(P\ S)|_{\Sigma}$. We now define U as follows: if $P = \{\}$, then also $X = \{\}$ and we define $U = Mod(F_p\langle Y\ \{\}\rangle)$; otherwise, we

define $U = P|_{\{p\}}$. Similarly, we define V as: if $S = \{\}$, then $V = Mod(F_p\langle X\ \{\}\rangle)$; otherwise $V = S|_{\{p\}}$. We now show that this $\binom{X\ Y}{U\ V}$ satisfies the desired properties.

First, we show that $\overline{\kappa}\binom{X\ Y}{U\ V} = (P\ S)$, i.e., that $\kappa\binom{X}{U} = P$ and $\kappa\binom{Y}{V} = S$. If $P = \{\}$, then $\kappa\binom{X}{U} = \kappa\binom{\{\}}{U} = \{\}$. If $P \neq \{\}$, then $X = P|_\Sigma$ and $U = P|_{\{p\}}$. Because $(P\ S)$ belongs to $\mathcal{D}_{T'}(\tilde{\mathcal{B}}_{\Sigma'}) \subseteq \overline{\kappa}(\tilde{\mathcal{B}}_{\Sigma'})$ (Proposition 4.6), we have that $P \in \kappa(\tilde{\mathcal{W}}_{\Sigma'})$. It follows that $P = \kappa\binom{P|_\Sigma}{P|_{\{p\}}}$ and, therefore, $P = \kappa\binom{X}{U}$. A similar argument shows that also $S = \kappa\binom{Y}{V}$. Second, we show that $\binom{X\ Y}{U\ V}$ is indeed a fixpoint of $\tilde{\mathcal{D}}_{T'}$. Let $\binom{X'\ Y'}{U'\ V'}$ be $\tilde{\mathcal{D}}_{T'}\binom{X\ Y}{U\ V}$. Because by construction $U, V \neq \{\}$, Proposition 4.8 implies that $\overline{\kappa}\binom{X'\ Y'}{U'\ V'} = \mathcal{D}_T(\overline{\kappa}\binom{X\ Y}{U\ V}) = \overline{\kappa}\binom{X\ Y}{U\ V}$, that is, $\kappa\binom{X'}{U'} = \kappa\binom{X}{U}$ and $\kappa\binom{Y'}{V'} = \kappa\binom{Y}{V}$. We now show that this implies $\binom{X}{U} = \binom{X'}{U'}$. Because both $U, U' \neq \{\}$, the equality $\kappa\binom{X'}{U'} = \kappa\binom{X}{U}$ implies that $X = X'$. Now, if $X, X' \neq \{\}$, then this equality also implies $U = U'$. If, on the other hand, $X = X' = \{\}$, then by construction, $U = Mod(F_p\langle Y\ \{\}\rangle) = U'$. We conclude that in both cases $\binom{X}{U} = \binom{X'}{U'}$. By a similar argument it follows that also $\binom{Y}{V} = \binom{Y'}{V'}$, so $\binom{X\ Y}{U\ V}$ is indeed a fixpoint of $\tilde{\mathcal{D}}_{T'}$. □

We now also examine the relation between stable fixpoints of $\mathcal{D}_{T'}$ and $\tilde{\mathcal{D}}_{T'}$. To this end, we compare the lower stable operators $C^{\downarrow}_{\mathcal{D}_{T'}}$ and $C^{\downarrow}_{\tilde{\mathcal{D}}_{T'}}$. It suffices to compare only these two operators, because, due to the symmetry of $\mathcal{D}_{T'}$ and $\tilde{\mathcal{D}}_{T'}$, we have that $C^{\uparrow}_{\mathcal{D}_{T'}} = C^{\downarrow}_{\mathcal{D}_{T'}}$ and $C^{\uparrow}_{\tilde{\mathcal{D}}_{T'}} = C^{\downarrow}_{\tilde{\mathcal{D}}_{T'}}$. Recall that $C^{\downarrow}_{\mathcal{D}_{T'}}$ is defined as mapping each S to *lfp*$([\mathcal{D}_T(\cdot, S)|)$ and, similarly, $C^{\downarrow}_{\tilde{\mathcal{D}}_{T'}}$ maps each $\binom{Y}{V}$ to *lfp*$([\tilde{\mathcal{D}}_{T'}\binom{\cdot\ Y}{\ V})|)$. By a straightforward induction over the construction of these least fixpoints, Proposition 4.8 now implies the following result.

Proposition 4.9. *For all* $\binom{X}{U} \in \mathcal{W}^{\downarrow c}_{\Sigma}$, $\kappa(C^{\downarrow}_{\tilde{\mathcal{D}}_{T'}}\binom{X}{U}) = C^{\downarrow}_{\mathcal{D}_{T'}}(\kappa\binom{X}{U})$.

This proposition now implies the following correspondence between stable fixpoints of $\mathcal{D}_{T'}$ and $\tilde{\mathcal{D}}_{T'}$, in precisely the same way as Theorem 4.10 follows from Proposition 4.8.

Theorem 4.11. *For all* $(P\ S) \in \mathcal{B}_\Sigma$, $(P\ S)$ *is a stable fixpoint (respectively, the well-founded fixpoint) of* $\mathcal{D}_{T'}$ *iff there exists a* $\binom{X\ Y}{U\ V} \in \tilde{\mathcal{B}}_{\Sigma'}$ *such that* $\overline{\kappa}\binom{X\ Y}{U\ V} = (P\ S)$ *and* $\binom{X\ Y}{U\ V}$ *is a stable fixpoint (the well-founded fixpoint) of* $\tilde{\mathcal{D}}_{T'}$.

Having shown that $\tilde{\mathcal{D}}_{T'}$ and $\mathcal{D}_{T'}$ have the same (stable) fixpoints, we can now proceed to relate the models of T' (under the various semantics we consider) to those of T, by using our algebraic theory of fixpoint extension to establish a correspondence between (stable) fixpoints of $\tilde{\mathcal{D}}_{T'}$ and $\mathcal{D}_T$.

$\tilde{\mathcal{D}}_{T'}$ is a fixpoint extension of $\mathcal{D}_T$

We now show that $\tilde{\mathcal{D}}_{T'}$ is indeed a fixpoint extension of $\mathcal{D}_T$. We first observe that, because $\mathcal{D}_{T'}$ is part-to-part constant, it is also part-to-part monotone. Therefore, all that

remains to be shown is that, for all $\binom{X\ Y}{U\ V}$ with $(U\ V) = \mathit{lfp}(\tilde{\mathcal{D}}_{T'}^{(X\ Y)})$, $\lceil \tilde{\mathcal{D}}_{T'}(\binom{X\ Y}{U\ V}) \rceil = \mathcal{D}_T(X\ Y)$. Of course, in this case, the condition that $(U\ V) = \mathit{lfp}(\tilde{\mathcal{D}}_{T'}^{(X\ Y)})$ is simply equivalent to $(U\ V)$ being the unique element in the image of $\tilde{\mathcal{D}}_{T'}^{(X\ Y)}$, i.e., to $\binom{X\ Y}{U\ V}$ belonging to $\tilde{\mathcal{B}}^*_{\Sigma'}$.

Proposition 4.10. *$\tilde{\mathcal{D}}_{T'}$ is a fixpoint extension of $\mathcal{D}_T$.*

Proof. We need to prove that for all $\binom{X\ Y}{U\ V} \in \tilde{\mathcal{B}}^*_{\Sigma'}$, $\lceil \tilde{\mathcal{D}}_{T'}(\binom{X\ Y}{U\ V}) \rceil = \mathcal{D}_T(X\ Y)$. By definition of these two operators, it suffices to show that, for all $\binom{X\ Y}{U\ V} \in \tilde{\mathcal{B}}^*_{\Sigma'}$, $F'\langle \overline{\kappa}(\binom{X\ Y}{U\ V})\rangle = F\langle X\ Y\rangle$ and $F'\langle \overline{\kappa}(\binom{Y\ X}{V\ U})\rangle = F\langle Y\ X\rangle$. Due to the symmetry of $\tilde{\mathcal{D}}_{T'}$, we have that $\binom{X\ Y}{U\ V} \in \tilde{\mathcal{B}}^*_{\Sigma'}$ iff $\binom{Y\ X}{V\ U} \in \tilde{\mathcal{B}}^*_{\Sigma'}$. Therefore, it suffices to show that for all $\binom{X\ Y}{U\ V} \in \tilde{\mathcal{B}}^*_{\Sigma'}$, $F'\langle \overline{\kappa}(\binom{X\ Y}{U\ V})\rangle = F\langle X\ Y\rangle$. Because the only difference between F' and F lies in the modal literals ψ' and ψ, it suffices to show that the way in which ψ' is evaluated in $F'\langle \overline{\kappa}(\binom{X\ Y}{U\ V})\rangle$ coincides with the way in which ψ is evaluated in $F\langle X\ Y\rangle$. The precise property that needs to be proven now depends on whether $\psi \in^+ F$ or $\psi \in^- F$, but by the same symmetry argument as above, we can cover both cases by showing that for all $\binom{X\ Y}{U\ V} \in \tilde{\mathcal{B}}^*_{\Sigma'}$, $\psi'\langle \overline{\kappa}(\binom{X\ Y}{U\ V})\rangle = \psi\langle X\ Y\rangle$.

Let us first consider the modal literal ψ' and let ρ' be the formula for which $\psi' = K\rho'$. For any belief pair $(P\ S)$, $K\rho'\langle P\ S\rangle$ is equal to the minimum of all truth values $\mathcal{H}_{I,(P\ S)}(\rho')$ for which $I \in P$. In the case of $(P\ S)$ being equal to $\overline{\kappa}(\binom{X\ Y}{U\ V})$ for some $\binom{X\ Y}{U\ V} \in \tilde{\mathcal{B}}^*_{\Sigma'}$, this is equal to the minimum m of all $\mathcal{H}_{I\cup J,(P\ S)}(\rho')$ for which $I \in X$ and $J \in U$. Moreover, because there is only one occurrence of the atom p in the entire formula ρ', there must exist a single "worst" truth value $\mathbf{v}$ for p which gives rise to this minimum; more formally put, for some $\mathbf{v}$, m is equal to the minimum of all $\mathcal{H}_{I\cup J,(P\ S)}(\rho'[p/\mathbf{v}])$ for which $I \in X$ and $J \in U$. Indeed, on the one hand, if $p \in^+ \rho'$, then $\mathbf{v}$ is the minimum of all $\mathcal{H}_{J,(\cdot\ \cdot)}(p)$ with $J \in U$. On the other hand, if $p \in^- \rho'$, then $\mathbf{v}$ is the maximum of all $\mathcal{H}_{J,(\cdot\ \cdot)}(p)$ with $J \in U$. Because $U, V \neq \{\}$ and the formula $\rho'[p/\mathbf{v}]$ no longer contains p, we now have that m is equal to the minimum of all $\mathcal{H}_{I,(X\ Y)}(\rho'[p/\mathbf{v}])$ for which $I \in X$. This is of course by definition equal to $\mathcal{H}_{\cdot,(X\ Y)}(K\rho'[p/\mathbf{v}]) = (\psi'[p/\mathbf{v}])\langle X\ Y\rangle$.

It now suffices to show that this $\mathbf{v}$ is equal to the way in which the modal literal $K\varphi$ is evaluated during the construction of $\psi\langle X\ Y\rangle$, i.e., that also $\psi\langle X\ Y\rangle = (\psi[K\varphi/\mathbf{v}])\langle X\ Y\rangle$. If $K\varphi \in^+ \psi$, then $K\varphi$ is evaluated in the belief pair $(X\ Y)$, i.e., it then suffices to show that $K\varphi\langle X\ Y\rangle = \mathbf{v} = \mathit{min}_{J\in U}(\mathcal{H}_{J,(\cdot\ \cdot)}(p))$. On the other hand, if $K\varphi \in^- \psi$, then $K\varphi$ is evaluated in the belief pair $(Y\ X)$, i.e., it then suffices to show that $K\varphi\langle Y\ X\rangle = \mathbf{v} = \mathit{max}_{J\in U}(\mathcal{H}_{J,(\cdot\ \cdot)}(p))$. It can now easily be checked from Proposition 4.7 that in both cases the needed equality holds.

□

We now have that, on the one hand, the (stable) fixpoints of $\mathcal{D}_{T'}$ coincide with those of $\tilde{\mathcal{D}}_{T'}$, while, on the other hand, the above proposition implies the correspondences between (stable) fixpoints of $\tilde{\mathcal{D}}_{T'}$ and $\mathcal{D}_T$, that were summarized in Section 4.2. This now allows us to relate the models of T' and T under various semantics.

Expansion, partial expansions and the Kripke-Kleene model

Because $\tilde{\mathcal{D}}_{T'}$ is part-to-part constant, Theorem 4.1 implies a correspondence between fixpoints of $\tilde{\mathcal{D}}_{T'}$ and $\mathcal{D}_T$. This gives the following result.

Theorem 4.12. *A belief pair $(P\ S) \in \mathcal{B}_\Sigma$ is a partial expansion (respectively, the Kripke-Kleene model) of T iff there exists a belief pair $(P'\ S') \in \mathcal{B}_{\Sigma'}$ such that $(P'\ S')|_\Sigma = (P\ S)$ and $(P'\ S')$ is a partial expansion (the Kripke-Kleene model) of T'.*

It is easy to see that for all $\binom{X\ Y}{U\ V} \in \tilde{\mathcal{B}}^*_{\Sigma'}$, $\binom{X\ Y}{U\ V}$ is exact iff $(X\ Y)$ is exact. Therefore, this correspondence between partial expansion also implies a correspondence between expansions.

Theorem 4.13. *A possible world structure $P \in \mathcal{W}_\Sigma$ is an expansion of T iff there exists a possible world structure $P' \in \mathcal{W}_{\Sigma'}$ such that $P'|_\Sigma = P$ and P' is an expansion of T'.*

Extensions, partial extensions and the well-founded model

As we recall from Section 4.2, to get a correspondence between the stable and well-founded fixpoints of our operators, we need an additional monotonicity property. Concretely, $\tilde{\mathcal{D}}_{T'}$ needs to be either part-to-whole or whole-to-part monotone. We now investigate when this is the case.

Proposition 4.11. *If $\psi \in^- F$, then $\tilde{\mathcal{D}}_{T'}$ is part-to-whole monotone.*

Proof. By symmetry of the operator $\tilde{\mathcal{D}}_{T'}$, it suffices to show that for all (X, Y) and $(U\ V) \leq (U'\ V')$, $\tilde{\mathcal{D}}^u_{T'}\binom{X\ Y}{U\ V} \leq \tilde{\mathcal{D}}^u_{T'}\binom{X\ Y}{U'\ V'}$. Furthermore, because $\tilde{\mathcal{D}}_{T'}$ is already know to be part-to-part monotone, it suffices to show that $Mod(F'\langle\overline{\kappa}\binom{X\ Y}{U\ V}\rangle) \supseteq Mod(F'\langle\overline{\kappa}\binom{X\ Y}{U'\ V'}\rangle)$. Because $\psi \in^- F$, this will be the case if $\psi'\langle\binom{Y}{V}\ \binom{X}{\cdot}\rangle \leq \psi'\langle\binom{Y}{V'}\ \binom{X}{\cdot}\rangle$. This now follows from $V \leq V'$. □

Proposition 4.12. *If $\psi \in^+ F$, $K\varphi \in^+ \psi$ and φ is an objective formula, then $\tilde{\mathcal{D}}_{T'}$ is whole-to-part monotone.*

Proof. By symmetry of $\tilde{\mathcal{D}}_{T'}$, it suffices to show that for all $\binom{X\ Y}{U\ V} \leq \binom{X'\ Y'}{U'\ V'}$, $\lfloor\tilde{\mathcal{D}}^u_{T'}\binom{X\ Y}{U\ V}\rfloor \leq \lfloor\tilde{\mathcal{D}}^u_{T'}\binom{X'\ Y'}{U'\ V'}\rfloor$. This is the case if $Mod(F_p\langle X\ Y\rangle) \supseteq Mod(F_p\langle X'\ Y'\rangle)$. Because $F_p = (K\varphi \Rightarrow p)$, this will be the case if $(K\varphi)\langle Y\ X\rangle \leq (K\varphi)\langle Y'\ X'\rangle$. Because φ is objective, $(K\varphi)\langle Y\ X\rangle$ depends only on Y and $(K\varphi)\langle Y'\ X'\rangle$ depends only on Y'. The fact that $Y \leq Y'$ now implies that $(K\varphi)\langle Y\ \cdot\rangle \leq (K\varphi)\langle Y'\ \cdot\rangle$. □

A summary of the monotonicity properties of $\tilde{\mathcal{D}}_{T'}$ can be found in Figure 4.2. By Theorem 4.11, we now obtain the following result.

Theorem 4.14. *If at least one of these conditions is satisfied:*

- $\psi \in^- F$ *or*

$\psi \in F$	$K\varphi \in \psi$	F_p	part-to-part	part-to-whole	whole-to-part
+	+	$K\varphi \Rightarrow p$	✓	×	✓*
-	+	$K\varphi \Rightarrow p$	✓	✓	✓*
+	-	$K\varphi \Leftarrow p$	✓	×	×
-	-	$K\varphi \Leftarrow p$	✓	✓	×

(*): Holds only if φ is objective.

Figure 4.2: Monotonicity properties of $\tilde{\mathcal{D}}_{T'}$.

- $K\varphi \in^{+} \psi$ *and* φ *is objective,*

then a belief $(P\ S) \in \mathcal{B}_\Sigma$ *is a partial extension (respectively, the well-founded model) of* T *iff there exists a belief pair* $(P'\ S') \in \mathcal{B}_{\Sigma'}$ *such that* $(P'\ S')|_\Sigma = (P\ S)$ *and* $(P'\ S')$ *is a partial extension (the well-founded model) of* T'*. Moreover, under the same condition, a possible world structure* $P \in \mathcal{W}_\Sigma$ *is an extension of* T *iff there exists a possible world structure* $P' \in \mathcal{W}_{\Sigma'}$ *such that* $P'|_\Sigma = P$ *and* P' *is an extension of* T'*.*

A final question that remains to be answered is what happens in the case where the above theorem is not applicable, i.e., when $\psi \in^{+} F$ and $K\varphi \in^{-} \psi$. It turns out that in this case there is no correspondence. We demonstrate this by the following example.

Example 4.4. Let T be the theory $\{K\neg Kq\}$. If we replace Kq by p, we get $T' = \{K\neg p; Kq \Leftarrow p\}$.

Let us first look at the well-founded model of T. We start by applying the stable operator $\mathcal{C}_{\mathcal{D}_T}$ to the least precise pair $(\mathcal{I}_\Sigma\ \{\})$. To obtain a new underestimate P', we construct $C^{\downarrow}_{\mathcal{D}_T}(\{\}) = \mathit{lfp}([\mathcal{D}_T(\cdot\ \{\})|)$. We find that $C^{\downarrow}_{T}(\{\}) = \mathcal{I}_\Sigma$, because:

$$[\mathcal{D}_T(\mathcal{I}_\Sigma, \{\})| = Mod(T\langle\{\}\ \mathcal{I}_\Sigma\rangle) = Mod(\mathbf{t}) = \mathcal{I}_\Sigma.$$

For the new overestimate, we have that $C^{\uparrow}_{\mathcal{D}_T}(\mathcal{I}_\Sigma) = \mathcal{I}_\Sigma$, because:

$$|\mathcal{D}_T(\mathcal{I}_\Sigma, \mathcal{I}_\Sigma)] = Mod(T\langle\mathcal{I}_\Sigma\ \mathcal{I}_\Sigma\rangle) = Mod(\mathbf{t}) = \mathcal{I}_\Sigma.$$

We therefore have that $\mathcal{C}_{\mathcal{D}_T}(\mathcal{I}_\Sigma\ \{\}) = (\mathcal{I}_\Sigma\ \mathcal{I}_\Sigma)$. Moreover, since by symmetry of $\mathcal{D}_T$, $C^{\downarrow}_{\mathcal{D}_T} = C^{\uparrow}_{\mathcal{D}_T}$, we now also see that $\mathcal{C}_{\mathcal{D}_T}(\mathcal{I}_\Sigma\ \mathcal{I}_\Sigma) = (\mathcal{I}_\Sigma\ \mathcal{I}_\Sigma)$. Therefore, this belief pair is the well-founded model of T, which is also its unique stable model.

We now perform a similar construction for T'. First, $C^{\downarrow}_{\mathcal{D}_{T'}}(\{\}) = \mathcal{I}_\Sigma$, because:

$$[\mathcal{D}_{T'}(\mathcal{I}_\Sigma, \{\})| = Mod(T\langle\{\}\ \mathcal{I}_\Sigma\rangle) = Mod(\mathbf{t}; \mathbf{t} \Leftarrow p) = \mathcal{I}_\Sigma.$$

Second, $C^{\uparrow}_{\mathcal{D}_{T'}}(\mathcal{I}_\Sigma) = \{\}$, as can be seen from the following computation:

$$|\mathcal{D}_{T'}(\mathcal{I}_\Sigma\ \mathcal{I}_\Sigma)] = Mod(\mathbf{f}; \mathbf{f} \Leftarrow p) = \{\}$$
$$|\mathcal{D}_{T'}(\mathcal{I}_\Sigma\ \{\})] = Mod(\mathbf{f}; \mathbf{f} \Leftarrow p) = \{\}$$

$\psi \in F$	$K\varphi \in \psi$	F_p	(part.) expansion	K-K	(part.) extension	wfm
+	+	$K\varphi \Rightarrow p$	✓*	✓*	✓*	✓*
-	+	$K\varphi \Rightarrow p$	✓	✓	✓	✓
+	-	$K\varphi \Leftarrow p$	✓	✓	×	×
-	-	$K\varphi \Leftarrow p$	✓	✓	✓	✓

(*): Holds only if φ is objective.

Figure 4.3: Semantics preserved by replacing $K\varphi$ by p.

Therefore, the well-founded model of T' is $(\mathcal{I}_{\{p,q\}}\ \{\})$. The restriction of this to the original alphabet Σ is $(\mathcal{I}_\Sigma\ \{\})$, which does not coincide with the well-founded model of T. Moreover, the partial stable models of T' are $(\mathcal{I}_{\{p,q\}}\ \{\})$, $(\{\}\ \mathcal{I}_{\{p,q\}})$ and $(\{\{q\},\{\}\}\ \{\{q\},\{\}\})$, which do not correspond to those of T either.

As a side note, we remark that if we were to ignore our analysis of Section 4.4.1 and take F_p to be the formula $K\varphi \Rightarrow p$ instead of $K\varphi \Leftarrow p$, then we would not get a correspondence either. Indeed, it can easily be checked that the well-founded model of $\{K\neg p; Kq \Leftarrow p\}$ is also $(\mathcal{I}_\Sigma\ \{\})$.

The results of our analysis of predicate introduction for autoepistemic logic can now be summarized by the table in Figure 4.3.

4.4.3 Discussion and related work

The nesting of modal operators is a source of computational complexity when evaluating autoepistemic theories in possible world structures, and therefore also when constructing models of such theories. Moreover, it also obscures the relation between this logic and other, related languages, such as logic programming and default logic. Indeed, both the Konolige transformation (Konolige 1988) from default logic into autoepistemic logic and, for instance, the transformations of logic programming into autoepistemic logic considered in (Bonatti 1995) map into the fragment without nested modal operators. In (Marek and Truszczyński 1991), a transformation is presented that, at least under the semantics of expansions, can reduce any theory to an equivalent one that does not have such nestings. This transformation preserves the original alphabet of the theory, but might lead to an exponential blow-up in its size, since it uses the standard propositional normalization technique of distributing disjunction over conjunction. Our results on predicate introduction can be used to achieve the opposite effect of avoiding such a blow-up, at the expense of an increase in the alphabet. A simple algorithm that does this, would be the following. As long as there are formulas of K-rank at least 2, select a formula $K\varphi$ with maximum K-rank and replace this by a new atom, in the way previously described. This algorithm reduces a theory T to a theory T'' without nested K operators, whose size is linear in the size of the original theory. Our results show that T' is equivalent to T on the original alphabet of T under the semantics of expansions, partial expansions, and Kripke-Kleene semantics. For the semantics of (partial) extensions and the well-founded semantics, this result does not

Logic programming		Autoepistemic logic		Preserves
Δ	Δ'	T	T'	stable fixpoints
$\{R \leftarrow \neg R\}$	$\left\{\begin{array}{l} R \leftarrow \neg P. \\ P \leftarrow R. \end{array}\right\}$	$\{KKr\}$	$\left\{\begin{array}{c} Kp \\ Kr \Rightarrow p \end{array}\right\}$	✓
$\{R \leftarrow \neg R\}$	$\left\{\begin{array}{l} R \leftarrow P. \\ P \leftarrow \neg R. \end{array}\right\}$	$\{\neg K \neg Kr\}$	$\left\{\begin{array}{c} \neg Kp \\ Kr \Leftarrow p \end{array}\right\}$	✓
$\{R \leftarrow R\}$	$\left\{\begin{array}{l} R \leftarrow P. \\ P \leftarrow R. \end{array}\right\}$	$\{\neg KKr\}$	$\left\{\begin{array}{c} \neg Kp \\ Kr \Rightarrow p \end{array}\right\}$	✓
$\{R \leftarrow R\}$	$\left\{\begin{array}{l} R \leftarrow \neg P. \\ P \leftarrow \neg R. \end{array}\right\}$	$\{K \neg Kr\}$	$\left\{\begin{array}{c} Kp \\ Kr \Leftarrow p \end{array}\right\}$	×

Figure 4.4: Correspondences between logic programming and autoepistemic logic.

quite hold. Indeed, here, our results do not give us a way of getting rid of nestings $K\varphi \in^- \psi$, where $\psi \in^+ F$. However, other nestings can still be eliminated.

Our analysis of the problem of predicate introduction in autoepistemic logic shows that our algebraic theorems also allow meaningful and useful results to be derived for this logic. Moreover, the algebraic concepts we have defined, i.e., those of fixpoint extension and part-to-part, part-to-whole, and whole-to-part monotonicity, have also proven to be useful analysis tools in this case. The use of these concepts reveals some interesting similarities to predicate introduction for logic programming, which might otherwise have gone unnoticed. Indeed, Figure 4.4 shows four cases of predicate introduction, in which, at the algebraic level, what happens in logic programming is precisely the same as what happens in autoepistemic logic. As such, the results of this section provide convincing evidence for the fact that our algebraic theory of fixpoint extensions is not only a convenient way of proving results for logic programming, but is also more widely applicable abstraction of a general knowledge representation principle.

4.5 Conclusion

In this chapter, we have developed a theory of fixpoint extension in the framework of approximation theory and studied two applications of these results. First, we investigated transformations for a general class of logic programming variants, under the supported model, Kripke-Kleene, stable, and well-founded model semantics. One of our most interesting results here was a general way of eliminating universal quantifiers from rule bodies under stable and well-founded semantics. Second, we also looked

at autoepistemic logic. Here, we studied a transformation to reduce the nesting depth of the modal operator K. We showed that, at the algebraic level, there are some remarkable parallels between the effects of this transformation and what happens in the case of logic programming. We were able to prove that this transformation preserves equivalence under the semantics of (partial) expansions and Kripke-Kleene semantics. Moreover, we also showed that, in a large number of cases, though not all, equivalence is also preserved under the well-founded semantics and the semantics of (partial) extensions.

In summary, we have demonstrated that our abstract concept of fixpoint extension can be used to analyze the problem of predicate introduction for different nonmonotonic logics and under different semantics. Moreover, this also exposes interesting relations between otherwise seemingly unrelated transformations. Together with the results of Chapter 3, this chapter therefore demonstrates that approximation theory is indeed a useful framework, in which properties of different logics with a fixpoint semantics can be studied in a clear, general and uniform way.

Part II

Constructive processes and causality

Chapter 5

Causal probabilistic logic

5.1 Introduction

In this chapter, we study the relation between constructive processes and the concept of *causality*. To motivate our interest in this topic, let us consider a causal statement of the following form:

Pneumonia causes chest pain. (5.1)

As a starting point, we present an analysis of the intuitive meaning of this statement. On the one hand, there is the rather obvious component that patients with pneumonia also suffer from chest pain. On the other hand, there are also two more subtle aspects of this statement.

- In essence, the causal statement is *dynamic* in nature, i.e., it refers to something that might happen, to a kind of occurrence, to an activation of some mechanism in the domain of discourse. In the case of statement (5.1), we are talking about a biological process involving viruses, lung tissue, nerves, and impulses. Once the patient has pneumonia, this mechanism is put into motion and eventually produces the phenomenon of chest pain. Because such a process might take some time to complete, this implies that if we were to observe pneumonia *now*, then the accompanying chest pain might only manifest itself *later*. This is in contrast to, for instance, the statement "pneumonia implies chest pain", which is static in nature and refers to a single point in time; i.e., if we observe pneumonia now, then the implied chest pain is also observed now. In what follows, we will refer to such mechanisms, to these "things that might happen", as *events*.

- Saying that pneumonia causes chest pain also implies that chest pain is a property that *needs* to be caused, i.e., the statement suggest that there exists some *a priori* state of affairs and that, in this state, chest pain is absent. It is only if the original state is affected in some way—for instance, if the mechanism by which pneumonia causes chest pain is activated—that the initial state of this property might change and the patient might suffer chest pain.

These observations suggest that if we have an exhaustive enumeration of all causal statements that are relevant with respect to some particular (aspect of a) domain of discourse, we could predict the final states in which this domain might end up by considering which possible sequences of events might occur. For instance, let us look at the following set of causal statements:

Pneumonia causes chest pain. (5.2)

Chest pain causes insomnia. (5.3)

Insomnia causes headache. (5.4)

Initially, the properties that might be caused (chest pain, insomnia, and headache) are all in their original state of being absent. If the patient now has pneumonia, then the event described by (5.2) will occur and cause chest pain, after which event (5.3) will also occur, causing insomnia, which finally also causes (5.4), thus resulting in a patient with a headache. On the other hand, for a patient without pneumonia, none of these events will be caused, so—if the enumeration really is exhaustive—there will be no headache.

Each of the causal statements we have considered so far describes an event whose effect is known with certainty. In real life, few events have this property. For instance, it is easy to image that the biological process underlying statement (5.1) (i.e., the virus infecting the lung tissue, which causes an nerve impulse to brain, which leads to the phenomenon of pain) might actually sometimes not result in the patient feeling chest pain at all. To take into account such non-determinism, we say, for instance:

Pneumonia *might* cause chest pain.

For another example, the following statement describes the possible effects of a risky surgical procedure:

> The surgery might cause the recovery of the patient, but it also might cause his death.

In both these cases, it is natural to quantify the uncertainty by assigning a probability to the possible outcome(s), such as:

Pneumonia causes chest pain with probability 0.8.

or:

> With probability 0.7, the surgery causes the recovery of the patient, but with probability 0.3, it causes his death.

Statements of this form will be our central topic. For obvious reasons, we will refer to such a statement as a *causal probabilistic event description*, or also, somewhat less accurately, as simply a *causal probabilistic event*, which we abbreviate as *CP-event*.

We will then consider a simple knowledge representation language, in which the causal structure of a domain is represented by an enumeration of all relevant CP-events. We call such a set of CP-events a *CP-theory* and refer to the language of all CP-theories

as *CP-logic*. As suggested by the above discussion, we will define the semantics of CP-logic by means of certain probabilistic processes, which can be constructed by considering the ways in which the events described by a CP-theory could actually happen. This generalizes in a rather straightforward way the process we described above in the context of the deterministic causal statements (5.2), (5.3) and (5.4); only, instead of each event having just a single effect, which results in a linear progression from an initial state to a unique final state, we now get events that can have multiple effects, which leads to a branching of possibilities, generating a tree-like progression from the initial state to a number of possible final states. We will call the probabilistic processes that can be generated by a given CP-theory its *execution models*.

In general, a CP-theory might have many execution models. Indeed, this is to be expected, since, clearly, certain relevant information about the dynamic evolution of a domain is not expressed in our language; in particular, a CP-theory does not incorporate any temporal information, i.e., it does not specify *when* a particular CP-event might happen, how long such an event will last, or even the order in which events happen. However, as we will show later, all execution models of a CP-theory generate precisely the same probability distribution over their final states. This uniqueness result is an interesting property, because, typically, we are not really interested in the actual details of the evolution of a domain anyway, but only care about the probability of arriving at a certain end result. Our result now shows that causal information, in the form of a CP-theory, suffices to know which possible outcome will occur with which probability. This offers an appealing explanation for why causality is such an important concept: causal information is in essence a compact and robust way of specifying just enough properties of the behaviour of a non-deterministic process to uniquely characterize the probability distribution that it generates.

CP-logic is essentially a causal probabilistic modelling language. It mainly distinguishes itself from other such languages by its explicit focus on the dynamic nature of causality—an aspect which is somewhat ignored in current literature. Indeed, let us consider, for instance, Pearl's influential work on causality (Pearl 2000). In this approach, the causal structure of a domain is described by a causal Bayesian network, i.e., a directed acyclic graph, in which every node corresponds to some random variable. The intuitive reading of such a network is that the value of every node is causally determined by the values of its parents in the graph. We can view such a network as an abstract representation of a class of probabilistic processes, in which, whenever the values of all parents of a node have been determined, an event occurs that propagates these values to the node itself. Here, too, we can make the observation that, in general, such a process is not not unique (because for any two nodes with no path between them, the network does not specify which of the events associated to these nodes will happen first), but that all these processes do generate the same distribution.

Now, for many domains, all of the relevant events might not fit directly into the rigid structure that a Bayesian network imposes. This will be the case, for instance, when more than one event is involved in determining the value of a single random variable, or when the propagation of values does not always happen in the same direction. In this sense, CP-logic extends Bayesian networks by allowing a more flexible and fine-grained representation of causal events, in which such phenomena can be modeled in a more direct and straightforward way. The uniqueness result described previously

shows that we can do this and still retain the property that every theory generates a single probability distribution over possible final states of the domain.

In summary, this chapter presents the following contributions. We explore the dynamic nature of causality and develop a representation language for causal knowledge, based on the construct of a causal probabilistic event. We prove that the information contained in a set of CP-events suffices to be able to predict the end result of the evolution of a domain. This study of causality extends and complements Pearl's work on this topic in a number of ways: we identify the concept of a CP-event as a unit of causal information, that is more basic than the parents-child relation underlying causal Bayesian networks. In this way, we get a more flexible and fine-grained representations of causal events, which allows more straightforward, compact, and elaboration tolerant models of causal knowledge. Moreover, we are able to support and clarify Pearl's claims on the stability of causal information and its importance for achieving compact representations, by showing that causal information captures precisely those aspects of the behaviour of a probabilistic process that are relevant for its final outcome.

This chapter is structured as follows. In Section 5.2, we formally define an initial, restricted version of CP-logic. In Section 5.3, we show how a certain kind of process can be modeled in this basic language, which suggests a way of defining a more general version of CP-logic. This will be done in Section 5.4. Section 5.5 then discussed the resulting definitions in more detail. In Section 5.6, we investigate the precise relation between CP-logic and Bayesian networks.

5.2 A logic of causal probabilistic events

In this section, we formally define the language of CP-logic. We assume that we have a logical vocabulary Σ at our disposal, such that any particular state of our domain of discourse corresponds to a Herbrand interpretation of Σ, i.e., a set of ground atoms. The restriction to Herbrand interpretations is made solely to ease notation; it is easy to extend all our definitions and results to arbitrary domains. We will make the standard distinction between properties that are endogenous (internal) to the process being modeled and properties that are exogenous (external); the endogenous properties are those which are affected by the process, while the exogenous properties simply describe the context in which it is taking place. To this end, we split the predicates of our vocabulary into a set of endogenous predicates and a set of exogenous ones.

5.2.1 Syntax

We want to describe the causes and effects of probabilistic events. A cause for an event will be represented by a first-order sentence φ, i.e., φ does not contain free variables. To represent the effects of an event, we will assume that our vocabulary is constructed in such a way that an event either does not affect the state of the domain at all, or causes a single (property corresponding to a) ground atom with an endogenous predicate to become true. A *CP-event* is then a statement of the following form:

$$(p_1 : \alpha_1) \vee \cdots \vee (p_n : \alpha_n) \leftarrow \varphi, \tag{5.5}$$

where φ is a first order sentence, the p_i are ground atoms with an endogenous predicate and the α_i are non-zero probabilities with $\sum \alpha_i \leq 1$. Such an expression is read as:

> "Property φ causes an event, whose effect is that at most one of the properties p_i becomes true, and for each p_i, the probability of it being caused is α_i."

If an event has a deterministic effect, i.e., it always causes some atom p with probability 1, we also write simply $p \leftarrow \varphi$ instead of $(p : 1) \leftarrow \varphi$. We allow the precondition φ to be absent, meaning that the CP-event always happens. In this case, the CP-event is called *unconditional* and we omit the '$\leftarrow$'-symbol as well.

A *CP-theory* is a finite multiset of CP-events. Throughout this section, we will restrict attention to CP-theories in which all sentences φ are positive formulas, i.e., they do not contain negation. Afterwards, Section 5.4 will examine how negation can be added to CP-logic.

Example 5.1. We consider a medical example. Pneumonia might cause angina with probability 0.2. Vice versa, angina might cause pneumonia with probability 0.3. A bacterial infection can cause either pneumonia (with probability 0.4) or angina (with probability 0.1). We consider bacterial infection as exogenous.

$$(Angina : 0.2) \leftarrow Pneumonia. \tag{5.6}$$

$$(Pneumonia : 0.3) \leftarrow Angina. \tag{5.7}$$

$$(Pneumonia : 0.4) \vee (Angina : 0.1) \leftarrow Infection. \tag{5.8}$$

We now define some notation to refer to different components of a CP-theory. The head $head(r)$ of a rule r of form (5.5) is the set of all pairs (p_i, α_i) appearing in the description of the effects of the event; the body $body(r)$ of r is its precondition φ. By $head_{At}(r)$ we denote the set of all ground atoms p_i for which there exists an α_i such that $(p_i, \alpha_i) \in head(r)$. Similarly, by $body_{At}(r)$ we will denote the set of all ground atoms p which "appear"[1] in $body(r)$. For the above example, if r is the CP-event (5.8), then $head(r) = \{(Pneumonia, 0.4), (Angina, 0.1)\}$, $head_{At} = \{Pneumonia, Angina\}$, $body(r) = Infection$ and $body_{At}(r) = \{Infection\}$.

We will call a CP-event $E \leftarrow \varphi$ a *rule* if we want to emphasize that we are referring to a syntactical construct. We also introduce the concept of a *non-ground* rule as a way of concisely representing sets of CP-events with identical structure. Concretely, such a non-ground rule is of the form:

$$\forall \mathbf{x}\ (A_1 : \alpha_1) \vee \cdots \vee (A_n : \alpha_n) \leftarrow \varphi,$$

where the atoms A_i and the formula φ now may contain free variables, taken from the universally quantified tuple of variables $\mathbf{x}$. Such a non-ground rule is seen as an abbreviation for the set of all rules $r[\mathbf{x}/\mathbf{t}]$ that result from replacing the variables $\mathbf{x}$ by a tuple $\mathbf{t}$ of ground terms in alphabet Σ. For instance, if we wanted to consider

[1] More formally, we use $body_{At}(r)$ to denote $B_F(body(r))$, where F is the Herbrand pre-interpretation and $B_F(\varphi)$ is the base we defined in Definition 3.14 of Section 3.3.2.

multiple people in the above example, we might include constants $\{John, Mary\}$ in our vocabulary Σ and write the non-ground rule

$$\forall x\ (Angina(x) : 0.2) \leftarrow Pneumonia(x),$$

to abbreviate the two CP-events

$$(Angina(John) : 0.2) \leftarrow Pneumonia(John).$$
$$(Angina(Mary) : 0.2) \leftarrow Pneumonia(Mary).$$

Because CP-theories are finite, the use of such abbreviations only makes sense in the context of a finite domain, i.e., when the vocabulary does not generate an infinite number of terms.

In our formal treatment of CP-logic, we will never consider non-ground rules, but always assume that these have already been expanded into a finite set of regular CP-events. When using such non-ground rules in examples, we will implicitly assume that predicates, functions and constants have been appropriately typed, in such a way as to avoid instantiations that are obviously not intended. In this way, we also allow ourselves to use function symbols, without immediately creating an infinite grounding.

5.2.2 Semantics

This section defines the formal semantics of CP-logic. A CP-theory expresses certain knowledge about the dynamic evolution of a domain. To make this more formal, we will assume that this evolution corresponds to a simple kind of probabilistic process, similar to, e.g., the processes considered in (Halpern and Tuttle 1993). Concretely, a process starts in some initial state and, through a sequence of possibly nondeterministic events, it probabilistically progresses towards any of a number of possible final states.

Our basic mathematical object will be that of a tree structure, in which the edges are labeled with probabilities. Each node in this tree corresponds to a state of the domain, with the root representing its initial state and the leaves its possible final states. Formally, we will assume a function $\mathcal{I}$ that maps each node s to a Herbrand interpretation $\mathcal{I}(s)$, which represents the state of the domain to which this node corresponds.

Definition 5.1. Let Σ be a vocabulary. A *probabilistic Σ-process* $\mathcal{T}$ is a pair $\langle T; \mathcal{I} \rangle$, such that:

- T is a tree structure, in which each edge is labeled with a probability, such that for every non-leaf node s, the probabilities of all edges leaving s sum up to precisely 1;
- $\mathcal{I}$ is a mapping from nodes of T to Herbrand interpretations of Σ.

If we interpret the probability associated to an edge (s, s') as the probability of making a transition from s to s' and assume that all these transitions are probabilistically independent, then we can associate to each node s the probability $\mathcal{P}(s)$ of a

random walk in the tree, starting from its root, passing through s. Indeed, for the root $\perp$ of the tree, $\mathcal{P}(\perp) = 1$ and for each other node s, $\mathcal{P}(s) = \prod_i \alpha_i$ where the α_i are all the probabilities associated to edges on the path from the $\perp$ to s. Essentially, the mapping $\mathcal{P}$ contains all the information that is present in the labeling of the edges and vice versa. To ease notation, we will sometimes take the liberty of identifying a probabilistic Σ-process $\langle T; \mathcal{I} \rangle$ with the triple $\langle T; \mathcal{I}; \mathcal{P} \rangle$ and ignoring the labels on the edges of T.

Each probabilistic Σ-process now induces an obvious probability distribution over the states in which the domain described by Σ might end up.

Definition 5.2. Let Σ be a vocabulary and $\mathcal{T} = \langle T; \mathcal{I}; \mathcal{P} \rangle$ a probabilistic Σ-process. By $\pi_{\mathcal{T}}$ we denote the probability distribution that assigns to each Herbrand interpretation I of Σ the probability $\sum_{s \in L_{\mathcal{T}}(I)} \mathcal{P}(s)$, where $L_{\mathcal{T}}(I)$ is the set of all leaves s of T for which $\mathcal{I}(s) = I$.

Like any probability distribution over interpretations, such a $\pi_{\mathcal{T}}$ also defines a set of possible worlds, namely that consisting of all I for which $\pi_{\mathcal{T}}(I) > 0$. If all the probabilities $\mathcal{P}(s)$ are non-zero, then this is simply the set of all $\mathcal{I}(l)$ for which l is a leaf of $\mathcal{T}$.

We now want to relate the transitions in such a probabilistic Σ-process to the events described by a CP-theory.

Definition 5.3. Let Σ be a vocabulary, C a CP-theory in this vocabulary and $\mathcal{T}$ a probabilistic Σ-process. Let $r \in C$ be a CP-event of the form:

$$(p_1 : \alpha_1) \vee \cdots \vee (p_n : \alpha_n) \leftarrow \varphi.$$

We say that r *happens* in a node s of $\mathcal{T}$ if s has $n+1$ children $s_1, \ldots, s_{n+1}$, such that:

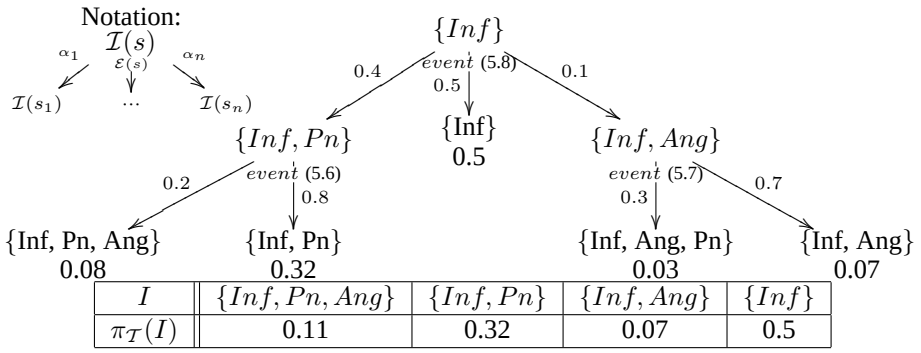
- For all $1 \leq i \leq n$, $\mathcal{I}(s_i) = \mathcal{I}(s) \cup \{p_i\}$ and the probability of edge (s, s_i) is α_i;
- For s_{n+1}, $\mathcal{I}(s_{n+1}) = \mathcal{I}(s)$ and the probability of the edge (s, s_{n+1}) is $1 - \sum_i \alpha_i$.

For simplicity, we will omit edges labeled with a probability of zero; this does not affect any of the following material.

This definition now allows us to link the transitions in a probabilistic Σ-process $\mathcal{T}$ to the events of a CP-theory C. Formally, we will consider a mapping $\mathcal{E}$ from each node s of $\mathcal{T}$ to an associated CP-event $r \in C$. Because, in our terminology, an event is something that happens at most once, the following definition will also consider, for a node s, the set of all events that have not yet happened in s, i.e., the set of all $r \in C$ for which there does not exist an ancestor s' of s such that $\mathcal{E}(s') = r$. We will denote this set as $\mathcal{R}_{\mathcal{E}}(s)$.

Definition 5.4 (Execution model–positive case). Let C be a positive CP-theory and X an interpretation of the exogenous predicates. A probabilistic Σ-process $\mathcal{T} = \langle T; \mathcal{I} \rangle$ is an *execution model* of C in context X, written $\mathcal{T} \models_X C$, iff there exists a mapping $\mathcal{E}$ from the non-leaf nodes of $\mathcal{T}$ to C, such that:

- For the root $\perp$ of $\mathcal{T}$, $\mathcal{I}(\perp) = X$;



I	$\{Inf, Pn, Ang\}$	$\{Inf, Pn\}$	$\{Inf, Ang\}$	$\{Inf\}$
$\pi_{\mathcal{T}}(I)$	0.11	0.32	0.07	0.5

Figure 5.1: A process $\mathcal{T}$ for Example 5.1 and its distribution $\pi_{\mathcal{T}}$.

- In each non-leaf node s, an event $\mathcal{E}(s) \in \mathcal{R}_{\mathcal{E}}(s)$ happens, such that its precondition is satisfied in s, i.e., $\mathcal{I}(s) \models body(\mathcal{E}(s))$;
- For each leaf l of $\mathcal{T}$, there are no events $r \in \mathcal{R}_{\mathcal{E}}(s)$ for which $\mathcal{I}(l) \models body(r)$.

If there are no exogenous predicates, we simply write $\mathcal{T} \models C$.

Example 5.1 has one execution model for every specific context X; the process for $X = \{Infected\}$ is depicted in Figure 5.1. In general, however, execution models are not unique. Let us illustrate this by another example.

Example 5.2. John and Mary are each holding a rock. John will throw his rock at a window. With probability 0.5, Mary will also throw her rock at this window. With probability 0.6, John's rock will hit the window, causing it to break, whereas Mary throwing her rock will cause the window to break with probability 0.8.

$$(Break : 0.8) \leftarrow Throws(Mary). \tag{5.9}$$

$$(Break : 0.6) \leftarrow Throws(John). \tag{5.10}$$

$$(Throws(Mary) : 0.5). \tag{5.11}$$

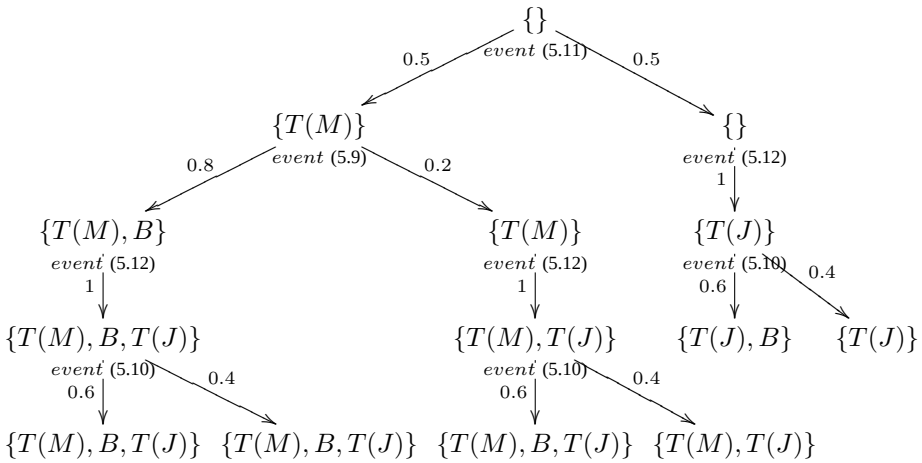
$$Throws(John). \tag{5.12}$$

This example has a number of different execution models. Two of these are depicted in Figure 5.2. We observe that, even though in these two processes events happen in a different order, they produce precisely the same probability distribution. This is a general property of positive CP-theories.

Theorem 5.1 (Uniqueness—positive case). *Let C be a positive CP-theory. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are both execution models of C, then $\pi_{\mathcal{T}_1} = \pi_{\mathcal{T}_2}$.*

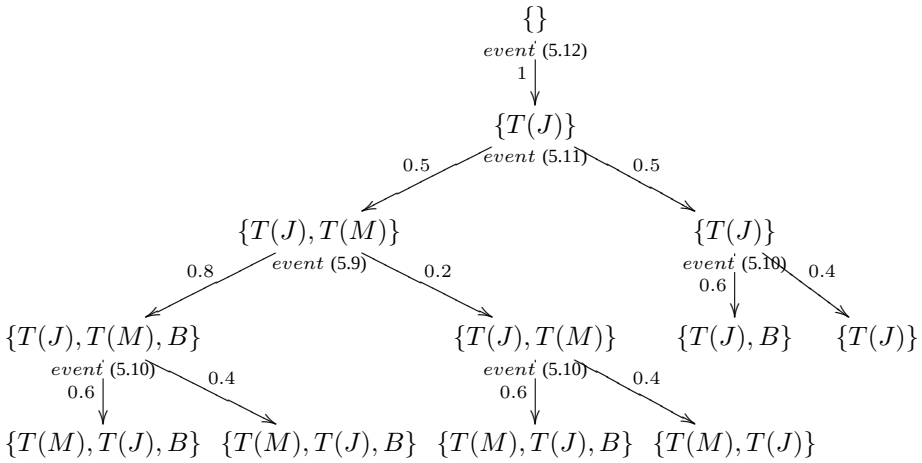
Proof. Proof of all the theorems in this chapter can be found in Chapter 6. □

As Example 5.2 illustrates, the causal information expressed by a CP-theory typically does not suffice to completely characterize a single probabilistic process, i.e., a



I	$\{T(M), T(J), B\}$	$\{T(M), T(J)\}$	$\{T(J), B\}$	$\{T(J)\}$
$\pi_{\mathcal{T}_1}(I)$	0.46	0.04	0.3	0.2

(a) $\mathcal{T}_1$: Mary throws—Mary's rock hits—John throws—John's rock hits.



I	$\{T(M), T(J), B\}$	$\{T(M), T(J)\}$	$\{T(J), B\}$	$\{T(J)\}$
$\pi_{\mathcal{T}_1}(I)$	0.46	0.04	0.3	0.2

(b) $\mathcal{T}_2$: John throws—Mary throws—Mary's rock hits—John's rock hits.

Figure 5.2: Two processes for Example 5.2.

CP-theory specifies some aspects of the behaviour of such a process, but not all. The above theorem now tells us that, as long as we are only interested in the end result of the process, all aspects that are not specified are actually irrelevant. This result is important for two reasons.

First, this result also suggests an appealing explanation for why causality is such a useful and important concept: causal information tells you just enough about the behaviour of a process to be able to predict its final outcome in every possible context, while allowing irrelevant details to be ignored. As such, it offers a compact and robust representation of the class of probability distributions that can result from such a process.

Second, in our construction of CP-logic, we have focused on the dynamic aspect of causality, which has motivated us to define the semantics of this language in terms of probabilistic processes. In this respect, CP-logic differs from the more common approach of, e.g., Bayesian networks, in which causal information is viewed as a description of a probability distribution over possible states of a domain. The above theorem relates these two views, because it allows us to not only view a CP-theory as describing a class of processes, but also as defining a unique probability distribution.

Definition 5.5. Let C be a CP-theory and X an interpretation for the exogenous predicates of C. By π_C^X, we denote the unique probability distribution $\pi_{\mathcal{T}}$, for which $\pi_{\mathcal{T}} \models_X C$. If there are no exogenous predicates, we simply write π_C.

A CP-theory can be viewed as mapping each interpretation for the exogenous predicates to a probability distributions over interpretations of the endogenous predicates or, to put it another way, as a conditional distribution over interpretations of the endogenous predicates, given an interpretation for the exogenous predicates.

Definition 5.6. Let C be a CP-theory and π a probability distribution over interpretations of all the predicates of C. π is a *model* of C, denoted $\pi \models C$ iff for each interpretation X of the exogenous predicates with $\pi(X) > 0$ and each interpretation J of the endogenous predicates, $\pi(J \mid X) = \pi_C^X(J)$.

If a CP-theory C has no exogenous predicates, then there is a unique π for which $\pi \models C$ and this is, of course, simply the distribution π_C.

Having defined this formal semantics for CP-logic, it is natural to ask how the causal interpretation that we have informally attached to expressions in our language is reflected in it. We see that every execution model of a CP-theory satisfies the following four properties, which seem to be fundamental principles of causal reasoning.

- The principle of *universal causation* states that an endogenous property can only be true if it has been caused by some event, i.e., all changes to the endogenous state of the domain must happen as the consequence of an event.
- The principle of *sufficient causation* states that if an event has a cause, then it must eventually occur.
- The principle of *no deus ex machina events* states that events do not happen spontaneously, i.e., an event can only occur if there is a cause for this and, moreover, events cannot cause themselves. This is a fundamental principle of causal reasoning, that goes back as far as Aristotle.

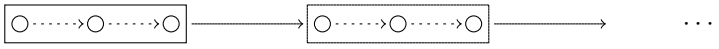


Figure 5.3: A global process as a sequence of local processes.

- The principle of *independent causation* states that every event affects the state of the world in a probabilistically independent way, i.e., knowing the outcome of one event does not give any information about the outcome of a different event. This principle ensures the modularity and robustness of the representation.

We now turn our attention to the question of whether we can extend the above definitions to the case where negation might appear in a CP-theory. However, this extension only makes sense in light of a certain modelling methodology for CP-logic, which we therefore first need to explain.

5.3 Modelling more complex processes in CP-logic

The kind of processes that we have been considering until now has been quite limited; in particular, the concept of *time* has been completely absent from them. For instance, when we spoke of "chest pain", this was a general, time-less property and not, say, chest pain at 9 a.m. on Monday morning. However, it is clear that being able to distinguish between the truth of the same property at different time points is often desirable. The following example illustrates this.

Example 5.3. A patient is admitted to hospital with pneumonia and stays there for a number of days. At each day, the pneumonia might cause him to suffer chest pain on that particular day with probability 0.6. With probability 0.8, a patient who has pneumonia on one day still has pneumonia the next day.

On the one hand, this example describes a progression through a sequence of days. On the other hand, for each day, it also describes an event that takes place entirely during this one particular day. In general, a process of this kind will look something like Figure 5.3: the global structure of the process is a succession between different time points and, at each particular time point, a local process might take place. In such a process, we can, therefore, distinguish two different kinds of events:

- There are events which propagate from one time point to the next; these make up the global structure of the process and are represented in Figure 5.3 by full-line arrows;
- There are events which take place entirely within in a single time point; these are part of some local process and are represented in Figure 5.3 by dotted-line arrows.

This raises the obvious question of how these more complicated processes relate to CP-logic. An important difference between modelling such processes and modelling

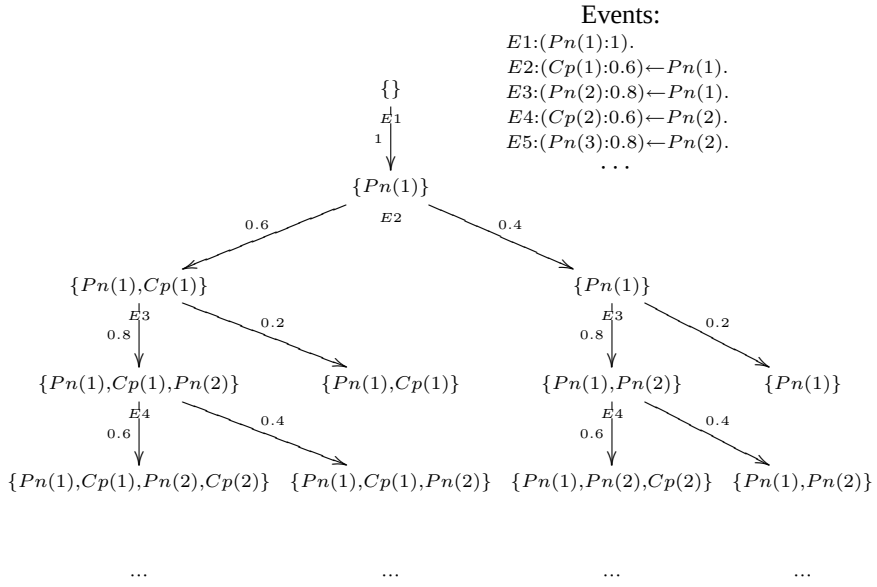


Figure 5.4: Initial segment of the intended model of Example 5.3.

the simpler processes of Section 5.2.2, is that we now need to distinguish between the values of properties at different time points, i.e., we can no longer represent every relevant property by a single ground atom, but instead we need a ground atom for every pair of such a property and a time point. For instance, to describe Example 5.3, we could construct an alphabet which has the following ground atoms:

- Referring to Day 1: $\{Pneumonia(1), Chestpain(1)\}$;
- Referring to Day 2: $\{Pneumonia(2), Chestpain(2)\}$;
- ...

Of course, it might be equally possible to use some other representation, such as $Pneumonia(Succ(Firstday))$ or $Pneumonia2$ instead of $Pneumonia(2)$. With the above alphabet, we can model Example 5.3 as follows:

$$Pneumonia(1). \tag{5.13}$$

$$\forall d\,(Pneumonia(d+1) : 0.8) \leftarrow Pneumonia(d). \tag{5.14}$$

$$\forall d\,(Chestpain(d) : 0.6) \leftarrow Pneumonia(d). \tag{5.15}$$

Here, the CP-events described by (5.14) are of the kind that propagate from one time point to a later time point, whereas (5.15) describes a class of "instantaneous" events, taking place inside of a single time point.

According to the informal description of Example 5.3, the intended model is the process shown in Figure 5.4. It can easily be seen that this is indeed an execution model

of the above CP-theory. We remark that this theory also has other execution models, which do not respect the proper ordering of time points, such as, e.g., the process in which all events that are instantiations of (5.14) happen before the instantiations of (5.15). However, since these "wrong" processes all generate the same probability distribution as the intended process anyway, this is harmless.

We also observe that, here, the correspondence between the states of the execution model and the actual states of the real world is less direct than it was in the examples of Section 5.2.2. Indeed, now, a state of an execution model not only describes the current state of the real world (as it did in Section 5.2.2), but also contains a trace of the entire history of the real world until that point.

Let us now make the above discussion more formal. We assume that, when constructing the alphabet Σ, we had in mind some function λ from the Herbrand base of Σ to an interval $[0..n] \subseteq \mathbb{N}$, such that, in our desired interpretation of this alphabet, each atom p refers to the state of some property at time point $\lambda(p)$. Typically, one would construct such an alphabet by adding explicit temporal arguments to predicates, as is done in, e.g., the event calculus or situation calculus. In the case of the above example, we had in mind the following λ:

- For each ground atom $Pneumonia(i)$, $\lambda(Pneumonia(i)) = i$;
- For each ground atom $Chestpain(i)$, $\lambda(Chestpain(i)) = i$.

If we now look again at the CP-events we wrote for this example, we observe that, whenever there is an atom in the head of a CP-event r that refers to the truth of some property at time i and an atom in the body of r that refers to the truth of some property at time j, it is the case that $i \geq j$. This is of course not a coincidence. Indeed, because, in the real world, causes precede effects, it should be impossible that the cause-effect propagation described by a CP-event goes backwards in time. We also remark that if the equality $i = j$ holds, the event is instantaneous (w.r.t. this particular granularity of time), i.e., it is one of those events that takes place entirely within a single time point.

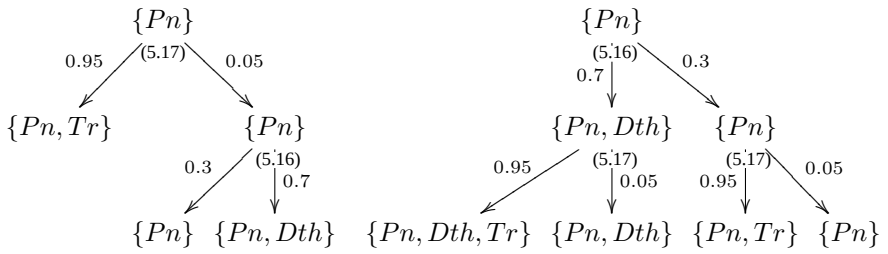
Definition 5.7. Let Σ be a vocabulary. A mapping λ from ground atoms of Σ to some interval $[0..n] \subseteq \mathbb{N}$ is a *temporal assignment* for a CP-theory C iff, for every $r \in C$, if $h \in head_{At}(r)$ and $b \in body_{At}(r)$, then $\lambda(h) \geq \lambda(b)$.

Such a temporal assignment λ also contains information about when events might happen. Concretely, if a CP-event r happens at time point i, then we would expect i to lie somewhere between the maximum of all $\lambda(b)$ for which $b \in body_{At}(r)$, and the minimum of all $\lambda(h)$ for which $h \in head_{At}(r)$. For a rule r, we write $t_\lambda(r)$ to denote this interval, i.e.,

$$t_\lambda(r) = [\max_{b \in body(r)} \lambda(b), \min_{h \in head_{At}(r)} \lambda(h)].$$

Now, if we are constructing a CP-theory with a particular temporal assignment λ in mind, then the process we are trying to model should be such that every CP-event r that actually happens does so at some time point $\kappa(r) \in t_\lambda(r)$. We remark that if an event r is instantaneous, then the interval $t_\lambda(r)$ will consists of a single time point and it is indeed clearly at this time point that the event should then happen.

A temporal assignment λ therefore imposes the following constraint on which processes can be considered reasonable.



(a) Sequence (5.17)–(5.16). (b) Sequence (5.16)–(5.17).

Figure 5.5: Two processes for Example 5.4.

Definition 5.8. Let C be a CP-theory with alphabet Σ and λ a temporal assignment for C. A mapping κ from r to $\mathbb{N}$ *respects* λ if for every CP-event r, $\kappa(r) \in t_\lambda(r)$. For such a κ, we say that a C-process $\mathcal{T}$ is a κ-process if events happen in the order imposed by κ, i.e., if for all successors s' of a node s, $\kappa(\mathcal{E}(s')) \geq \kappa(\mathcal{E}(s))$. Finally, a C-process $\mathcal{T}$ *follows* λ if there exists a κ that respects λ and for which $\mathcal{T}$ is a κ-process.

It can now be shown that for any CP-theory C and any temporal assignment λ for C, C will have an execution model that follows λ.

Theorem 5.2. *Let C be a CP-theory with a temporal assignment λ. There exists an execution model $\mathcal{T}$ of C, such that $\mathcal{T}$ follows λ.*

This result shows that if we construct a CP-theory C with a particular temporal assignment in mind, then C will have an execution model in which the events happen in precisely the order dictated by this temporal assignment. Therefore, the modelling methodology that we have suggested in this section is indeed valid. In the case of Example 5.3, the process shown in Figure 5.4 is an execution model that follows the temporal assignment λ specified above.

5.4 CP-logic with negation

So far, we have only considered positive formulas as preconditions to events. In this section, we examine whether it is possible to relax this requirement. We first consider a small example.

Example 5.4. Having pneumonia causes a patient to receive treatment with probability 0.95. Untreated pneumonia may cause death with probability 0.7.

$$(Death : 0.7) \leftarrow Pneumonia \wedge \neg Treatment. \tag{5.16}$$

$$(Treatment : 0.95) \leftarrow Pneumonia. \tag{5.17}$$

Figure 5.5 shows two processes for this example that satisfy all the requirements that we previously imposed for positive theories. It is obvious, however, that in this

case the difference between these two processes does affect the final outcome, since the probability of survival is much better in the first process. So, simply including negation in this naive way would give rise to ambiguities, causing our desirable uniqueness property (Theorem 5.1) to be lost. At first sight, this might suggest that we should forget about negation altogether. However, it turns out that it is possible to do something better.

The key idea is the distinction made earlier between two different kinds of events: instantaneous events that take place entirely within a single time point and events that propagate from one time point to a later one. Now, if negation appears in the description of an event of the first kind, then this is truly ambiguous and there is no way of singling out some intended interpretation. However, the situation is different when negation appears in the description of the second kind of event. Indeed, let us assume, for instance, that (5.16) is such an event. This means that if $Treatment$ is a property whose truth gets decided at a certain point in time (for instance, when the patient is first admitted to hospital), then $Death$ is a property that describes the condition of the patient at some strictly later time (for instance, after she has been in hospital for a week). Under this interpretation, it is clear that patient cannot die before she has had the chance to receive treatment. So, from our assumption, it directly follows that whichever events make up the part of the process that determines whether the patient receives treatment, should take place before event (5.16), i.e., in this case, process 5.5(a) is right and process 5.5(b) is wrong.

Our approach to including negation into CP-logic is now that we will simply exclude the ambiguous case from consideration, i.e., we prohibit the use of negation in instantaneous events. We can then strengthen the concept of an execution model, as we first defined it for positive theories, in such a way that it produces the correct result for the remaining theories. In other words, we will adapt our semantics in such a way that, for instance, a CP-event of the form:

$$(P : 0.5) \leftarrow \neg Q$$

will be interpreted as though Q is an atom that refers to some property at time point i and P refers to some property at time j with $j > i$. Ultimately, it is then of course the responsibility of the user to make sure that this interpretation corresponds to her intentions. Essentially, this boils down to ensuring that the vocabulary is built using a sufficiently fine-grained notion of time.

To formally extend our semantics in this way, we need some more mathematical machinery. The basic idea is that, before we allow an event to happen, we should make sure that our process has actually already progressed far enough along; to be more concrete, if we have an event r whose precondition contains some negative literal $\neg A$ and A is an atom that might be caused at time point i, then r should, at the earliest, happen at time point $i + 1$. Of course, CP-logic does not explicitly incorporate any information about time points, which makes it not completely obvious how we should incorporate this idea into our semantics. However, we can nevertheless extract the wanted information from a CP-theory, by considering whether the current truth value of A is final or whether it is still subject to change. Indeed, if the process is currently in a state where A is false, but it is still possible that an event happens which causes A,

then we must be in some time point j for which $j \leq i$ and, therefore, it is still too early for the event r to happen.

Formalizing this idea requires us to take into account the possibility that, in some particular state, the truth of a formula is still "in progress," i.e., that it is still unknown whether it will eventually hold or not. This brings us naturally to three valued logic, where, as we already saw in Chapter 2, we have truth values **t** (already certainly true), **f** (already certainly false), and **u** (still unknown). Recall that a three-valued interpretation ν is a mapping from the ground atoms of our vocabulary to the set of truth values $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$, which induces for each formula φ a truth value φ^ν.

Now, if our probabilistic process is in a state s, then the atoms of which we are already sure that they are true are precisely those in $\mathcal{I}(s)$. To figure out which atoms are still unknown, we need to look at which events might still happen, i.e., at those rules r, for which $body(r)^\nu \neq \mathbf{f}$. Whenever we find such a rule, we know that the atoms in $head(r)$ might all still be caused and, as such, they must be at least unknown. We will now look at a derivation sequence, in which we start by assuming that everything that is currently not **t** is **f** and then gradually build up the set of unknown atoms by applying this principle. Formally, we consider a sequence $(\nu_i)_{0 \leq i \leq n}$ of three-valued interpretations. Initially, ν_0 assigns **f** to all atoms not in $\mathcal{I}(s)$. For each $i > 0$, there must be a rule r with $body(r)^{\nu_i} \neq \mathbf{f}$, such that, for all $p \in head(r)$ with $\nu_i(p) = \mathbf{f}$, it is the case that $\nu_{i+1}(p) = \mathbf{u}$, while for all other atoms p, $\nu_{i+1}(p) = \nu_i(p)$. A sequence that satisfies these properties is called a *hypothetical derivation sequence* in s. Such a sequence is *terminal* if it cannot be extended. A crucial property is now that all such sequences reach the same limit.

Theorem 5.3. *Every terminal hypothetical derivation sequence reaches the same limit, i.e., if $(\nu_i)_{0 \leq i \leq n}$ and $(\nu'_i)_{0 \leq i \leq m}$ are such sequences, then $\nu_n = \nu'_m$.*

For a state s in a probabilistic process, we will denote this unique limit as ν_s. Such a ν_s now provides us with an estimate of which atoms might still be caused, given that we are already in state s. We can now tell whether the part of the process that determines the truth of a formula φ has already finished by looking at ν_s; indeed. we can consider this process to be finished iff $\varphi^{\nu_s} \neq \mathbf{u}$. We now extend the concept of an execution model to arbitrary CP-theories as follows.

Definition 5.9. Let C be a CP-theory in alphabet Σ, $\mathcal{T}$ a probabilistic Σ-process, and X an interpretation of the exogenous predicates of C. $\mathcal{T}$ is an *execution model* of C in context X iff

- $\mathcal{T}$ satisfies the conditions of Definition 5.4 (execution model–positive case);
- For every node s, $(body(\mathcal{E}(s))^{\nu_s} \neq \mathbf{u}$.

In the case of Example 5.4, this indeed gives us the result described above, i.e., the process in Figure 5.5(a) is an execution model of the example, while the one in Figure 5.5(b) is not. Indeed, if we look at the root $\perp$ of this tree, with $\mathcal{I}(\perp) = \{Pneumonia\}$, we see that we can construct the following terminal hypothetical derivation sequence:

- ν_0 assigns **f** to $Treatment$ and $Death$;

- ν_1 assigns **u** to $Treatment$;
- ν_2 assigns **u** to $Death$, because $(\neg Treatment \wedge Pneumonia)^{\nu_1} = \mathbf{u}$;

As such, the only event that can initially happen is the one by which the patient might receive treatment. Afterwards, in every descendant s of $\bot$, $\nu_s(Treatment)$ will be either $\mathbf{t}$ of $\mathbf{f}$. In the branch where it is $\mathbf{f}$, the event by which the patient dies of untreated pneumonia will subsequently happen.

In Section 5.2.2, we isolated a number of important principles from our definition of the semantics of CP-logics. In a similar way, the extra condition that we have imposed in this section can be formulated as follows.

- The *principle of temporal precedence* states that, whenever a property φ might cause an event E, then the part of the process that is involved in determining the truth of φ happens *before* the event E itself can happen.

Under the assumptions we have been making in this section—namely, that atoms are linked to time points in such a way that negation only occurs in events where there is a propagation from one time point to a later one—-this principle can be motivated by the fundamental property of the physical world that a cause must always precede its effects.

5.5 Discussion

We now check whether the way in which the previous section has extended the concept of an execution model to cope with negation indeed satisfies the goals that we originally stated.

5.5.1 The case of positive theories

First of all, we remark that, for positive CP-theories, the new definition (Def. 5.9) simply coincides with the original one (Def. 5.4). Indeed, because, according to our original definition, it must be the case that $\mathcal{I}(s) \models \mathcal{E}(s)$ for each non-leaf node s, this is an immediate consequence of the following trivial theorem.

Theorem 5.4. *Let s be a node in a probabilistic Σ-process. For any positive formula φ, if $\mathcal{I}(s) \models \varphi$, then $\nu_s(\varphi) = \mathbf{t}$.*

Proof. Throughout a hypothetical derivation sequence, the truth of an atom p can only increase; in particular, if $\nu_i(p) = \mathbf{t}$, then $\nu_{i+1}(p) = \mathbf{t}$. The theorem therefore follows by induction from the fact that, by definition, $\mathcal{I}(s) \models \varphi$ implies $\varphi^{\nu_0} = \mathbf{t}$. □

We conclude that, for positive CP-theories, the new definition is simply equivalent to the old one.

5.5.2 Uniqueness theorem regained

Second, the uniqueness theorem now indeed extends beyond positive theories.

Theorem 5.5 (Uniqueness—general case). *Let C be a CP-theory and X an interpretation of the exogenous predicates of C. If $\mathcal{T}$ and $\mathcal{T}'$ are execution models of C in context X, i.e., $\mathcal{T} \models_X C$ and $\mathcal{T}' \models_X C$, then $\pi_{\mathcal{T}} = \pi_{\mathcal{T}'}$.*

Proof of this theorem will again be provided in Chapter 6,

5.5.3 Events can happen in the right order

Third, we will now show that it is indeed the case that the intended models of a theory—i.e., those processes in which the progression of time points happens in the right order—satisfy the additional condition that we have imposed. To formalize our assumption that negation does not appear in instantaneous events, we need to make a distinction between those atoms from some $body_{At}(r)$ that appear only in a *positive* context and those which occur at least once in a *negative* context. The set of all the latter atoms will be denoted as $body^{-}_{At}(r)$, whereas that of all the former ones is $body^{+}_{At}(r)$[2]. Using this notation, we can now formalize our assumption as follows.

Definition 5.10. A CP-theory C is *stratified* if there exists a mapping λ from its Herbrand base to $\mathbb{N}$, such that, for all ground atoms h and b:

- If there is CP-event r with $h \in head_{At}(r)$ and $b \in body^{+}_{At}(r)$, then $\lambda(h) \geq \lambda(b)$;
- If there is CP-event r with $h \in head_{At}(r)$ and $b \in body^{-}_{At}(r)$, then $\lambda(h) > \lambda(b)$;

We remark that, in particular, all positive theories are stratified, because, for such a theory, we can simply assign the same number to all ground atoms. The following is an example of a more complex stratified theory.

Example 5.5. We consider a time line divided into a number of different time slots, as illustrated in Figure 5.6. In the first time slot, a client sends a request to a server. If the server receives a request, then with probability 0.5, he accepts it and sends a reply, all within the same time slot as that in which he received the request. If the client has sent a request and has not received a reply at the end of the time slot, he will repeat his request. A message that is sent has a probability of 0.8 of reaching the recipient in the

[2] Formally, we define, for all sentences φ, the sets $At^{+}(\varphi)$ and $At^{-}(\varphi)$ by simultaneous induction as:

- For $p(\mathbf{t})$ a ground atom, $At^{-}(p(\mathbf{t})) = \{\}$ and $At^{+}(p(\mathbf{t})) = \{p(\mathbf{t})\}$;
- For $\varphi \circ \psi$, with $\circ$ either $\vee$ or $\wedge$, $At^{+}(\varphi \circ \psi) = At^{+}(\varphi) \cup At^{+}(\psi)$ and $At^{-}(\varphi \circ \psi) = At^{-}(\varphi) \cup At^{-}(\psi)$;
- For $\neg\varphi$, $At^{+}(\neg\varphi) = At^{-}(\varphi)$ and $At^{-}(\neg\varphi) = At^{+}(\varphi)$;
- For $\Theta x\ \varphi$, with Θ either $\forall$ or $\exists$, $At^{+}(\Theta x\ \varphi) = \cup_{t \in H_U(\Sigma)} At^{+}(\varphi[x/t])$ and $At^{-}(\Theta x\ \varphi) = \cup_{t \in H_U(\Sigma)} At^{-}(\varphi[x/t])$, where $H_U(\Sigma)$ is the Herbrand universe.

We can then define $body^{-}_{At}(r) = At^{-}(body(r))$ and $body^{+}_{At}(r) = body_{At}(r) \setminus body^{-}_{At}(r)$.

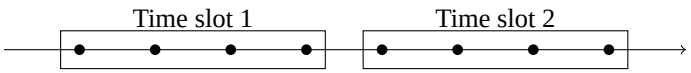


Figure 5.6: A division into time slots.

same time slot as it was sent; with probability 0.1, it reaches the recipient only in the next slot; with the remaining probability of 0.1, it will be lost.

$$(Send(Client, Req, Server, 1) : 0.7). \tag{5.18}$$

$$\forall t\ (Accept(t) : 0.5) \vee (Reject(t) : 0.5) \leftarrow Recvs(Server, Req, t). \tag{5.19}$$

$$\forall t\ Send(Server, Answer, Client, t) \leftarrow Accept(t). \tag{5.20}$$

$$\forall t, s, r, m\ (Recvs(r, m, t) : 0.8) \vee (Recvs(r, m, t+1) : 0.1) \leftarrow Send(s, m, r, t). \tag{5.21}$$

$$\forall t\ Send(Client, Req, Server, t+1) \leftarrow Send(Client, Req, Server, t) \wedge \neg Recvs(Client, Answer, t). \tag{5.22}$$

In this CP-theory, the events described by (5.18), (5.19) and (5.20) all take place inside of a single time slot; the events described by (5.21) might either take place within one time slot or constitute a propagation to a later time slot, depending on which of the possible effects actually occurs; finally, the events described by (5.22) all propagate to a later time slot. Because these last events are the only ones in which negation occurs, this theory is indeed stratified with respect to the obvious function λ that maps each ground atom to the temporal argument appearing in it. Because the theory therefore satisfies our assumption, we would expect our semantics to give the intended result. The following theorem shows that this is indeed the case.

Theorem 5.6. *Let C be a CP-theory which has a stratification λ. Every probabilistic Σ-process that follows λ and satisfies the original conditions of Definition 5.4 also satisfies the additional condition imposed by Definition 5.9 and is, therefore, an execution model of C. Moreover, such a process always exists.*

This theorem shows that if we construct a CP-theory with a given temporal assignment in mind, and make sure that the assumption about the use of negation is satisfied, then this theory will have an execution model in which the events happen in the intended order.

We remark that non-stratified theories do not always have an execution model. Let us illustrate this by the following example.

Example 5.6. A game is being played between two players, called $White$ and $Black$. If $White$ does not win, this causes $Black$ to win and if $Black$ does not win, this causes

$White$ to win.

$$Win(White) \leftarrow \neg Win(Black).$$
$$Win(Black) \leftarrow \neg Win(White).$$

This CP-theory does not have an execution model. Indeed, for the root $\perp$ of any such model, it would be the case that $\mathcal{I}(\perp) = \{\}$, which means that the bodies of both rules are satisfied, so, by the principle of sufficient causation, one of these events should happen. However, it is also the case that $(\neg Win(White))^{\nu_\perp} = \mathbf{u} = (\neg Win(Black))^{\nu_\perp}$, so neither event can happen.

If a CP-theory has no execution models, then this means that it either violates our assumption about the use of negation, or simply represents an incorrect or incomplete model of the domain in question. Indeed, it is clear that, for instance, the above example does not suffice to describe a game that could be played in practice; obviously one of the two players should win, but there is no way of deciding which.

Theories which have no execution models are obviously not of interest.

Definition 5.11. A CP-theory C is *valid* in an interpretation X for its exogenous predicates if it has at least one execution model in context X. If C is valid in all contexts X, we simply say that C is valid.

Clearly, it is only if C is a valid CP-theory, that we can associate a probability distribution π_C to it.

This discussion raises the question of whether there are actually *any* reasonable CP-theories that are not stratified. To answer this question, we observe that the existence of a stratification is a purely syntactical concept and there might be theories which are, intuitively or semantically speaking, "stratified" in some sense, but not syntactically so. For instance, we could consider the following example.

Example 5.7. We consider a rather trivial game, in which there is a stack of two objects. In every turn, a player may remove either one or two of these objects. The player to take the last object wins. Let us assume that this game is played between two players A and B, who both make random moves, and that these players flip a coin to decide who goes first. We can model this game by the following (rather contrived) set of CP-events, which essentially states that the player who starts gets a chance to win the game (by taking both objects), but if he does not take this chance, then the other player will win (by making the only available move of taking the last remaining object).

$$(Starts(A) : 0.5) \vee (Starts(B) : 0.5). \tag{5.23}$$
$$(Win(A) : 0.5) \leftarrow Starts(A). \tag{5.24}$$
$$(Win(B) : 0.5) \leftarrow Starts(B). \tag{5.25}$$
$$Win(A) \leftarrow Starts(B) \wedge \neg Win(B). \tag{5.26}$$
$$Win(B) \leftarrow Starts(A) \wedge \neg Win(A). \tag{5.27}$$

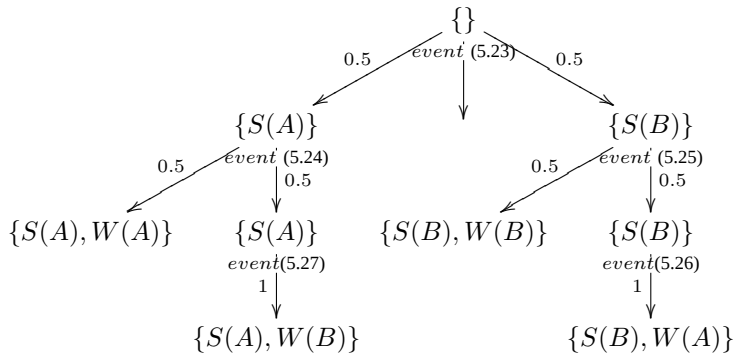


Figure 5.7: An execution model for Example 5.7.

Even though, technically speaking, this example is not stratified, it is clear that, once it has been decided which player will start, the remaining game (i.e., the set of rules in which the right player starts) does satisfy the assumption that all events containing negation propagate from one time point (the first move) to a later time point (the second move). Such examples, which do not admit a static, syntactical stratification, but which do have this kind of dynamic, semantical "stratification", can also be handled by our semantics. Indeed, Figure 5.7 shows an execution model for the above example.

5.5.4 The representation of time in CP-logic

In the preceding sections, we have encountered two quite different styles of knowledge representation, namely that in which the vocabulary explicitly includes time and that in which it does not. The first style is perhaps more typical of logic-based languages—we have already mentioned situation and event calculus in this respect—whereas the second kind seems to be more common for, e.g., Bayesian networks. Both styles are useful and natural ways of thinking about causal events. It is therefore an interesting feature of CP-logic that it allows both to be used, possibly even within the same theory, i.e., it might be perfectly reasonable to make time explicit for only certain properties or certain time points.

On the one hand, abstracting away time often leads to significantly smaller and simpler representations. On the other hand, theories with explicit time also have certain advantages: for instance, it might be easier to prove their correctness with respect to a given specification or they might be more robust with respect to future changes. The decision whether to make time explicit—and to what extent—is partly up to the intuitions and taste of the knowledge engineer. There are, however, also some objective indications that explicit time might be needed.

First of all, it might be the case that time already plays a significant role in the problem description itself. For instance, a person who has contracted a contagious disease might infect a person they come into contact with, but only if the contact occurs *after* the initial infection. If such properties are relevant to the situation being modeled, then

some form of temporal arguments will be necessary. Second, in a theory where time is implicit (or not sufficiently fine-grained), negation might appear in instantaneous events, which could yield undesirable results. By including explicit temporal arguments (of an appropriate granularity), this problem can be solved. Third, the semantics of CP-logic assumes that each ground atom represents a property that starts out being false and might become true. Therefore, if it is necessary to handle properties whose truth value might change more than once—for instance, some movable object might start out at rest, then move for a while, but eventually halt again—then at least as much temporal detail must be introduced as is needed to distinguish between the different states of this property.

5.6 The relation to Bayesian networks

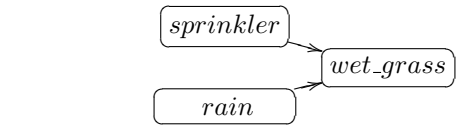
In this section, we investigate the relation between CP-logic and Bayesian networks. Because CP-logic is meant to offer a more fine-grained and flexible representation of causal events than Bayesian networks, we would expect our analysis to show the following. First of all, if all the events in some domain happen to fit directly into the structure imposed by a Bayesian network, then the representation of this domain should be essentially the same in both formalisms. Second, if this is not the case, then we would expect CP-logic to offer some representational advantages.

Before we begin, let us briefly recall the definition of a Bayesian network. Such a network consists of a directed acyclic graph and a number of probability tables. Every node n in the graph represents a random variable, which has some domain $dom(n)$ of possible values. A network B defines a unique probability distribution π_B over the set of all possible assignments $n_1 = v_1, \ldots, n_m = v_m$ of values to all of these random variables, with all $v_i \in dom(n_i)$. First, this π_B must obey a probabilistic independence assumption expressed by the graph, namely, that every node n is probabilistically independent of all of its non-descendants, given its parents. This allows the probability $\pi_B(n_1 = v_1, \ldots, n_m = v_m)$ of such an assignment of values to all random variables to be rewritten as a product of conditional probabilities $\prod_i \pi_B(n_i = v_i \mid pa(n_i) = \mathbf{v})$, where each $pa(n_i)$ is the tuple of all parents of n_i in the graph. The probability tables associated to the network now specify precisely all of these conditional probabilities $\pi_B(n_i = v_i \mid pa(n_i) = \mathbf{v})$. The second condition imposed on π_B is then simply that all of these conditional probabilities must match the corresponding entries in these tables. It can be shown that this indeed suffices to uniquely characterize a single distribution.

5.6.1 Bayesian networks in CP-logic

As already explained in the introduction, a Bayesian network can also be seen as a description of a class of probabilistic processes. We now first make this more precise. To make it easier to compare to CP-logic later on, we will start by introducing a logical vocabulary for describing a Bayesian network.

Definition 5.12. Let B be a Bayesian network. The vocabulary Σ_B consists of a predicate symbol P_n for each node n of B and a constant C_v for each value v in the domain of n.



	spr, rain	spr,¬rain	¬spr, rain	¬spr,¬rain
wet	0.98	0.8	0.9	0

sprinkler	0.2

rain	0.4

Figure 5.8: Bayesian network for the sprinkler example.

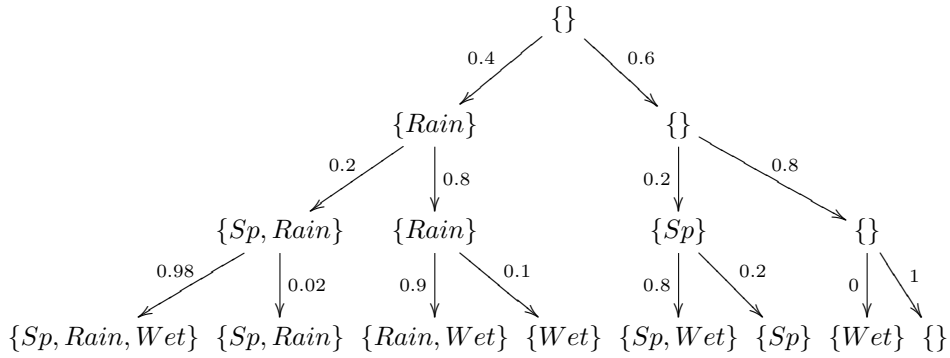


Figure 5.9: Process corresponding to the sprinkler Bayesian network.

Now, we want to relate a Bayesian network B to a class of Σ_B-processes. Intuitively, we are interested in those processes, where the flow of events follows the structure of the graph and every event propagates the values of the parents of a node to this node itself. We illustrate this by the following famous example.

Example 5.8 (Sprinkler). The grass can be wet because it has rained or because the sprinkler was on. The probability of the sprinkler causing the grass to be wet is 0.8; the probability of rain causing the grass to be wet is 0.9. The *a priori* probability of rain is 0.4 and that of the sprinkler having been on is 0.2.

The Bayesian network formalization of this example can be seen in Figure 5.8. Figure 5.9 shows a process that corresponds to this network. Here, we have exploited the fact that all random variables of the Bayesian network are boolean, by representing every random variable by a single atom, i.e., writing for instance Wet and $\neg Wet$ instead of $Wet(True)$ and $Wet(False)$. Formally, we define the following class of processes for a Bayesian network.

Definition 5.13. Let B be a Bayesian network. A *B-process* is a probabilistic Σ_B-process $\mathcal{T}$ for which there exists a mapping $\mathcal{N}$ from nodes of $\mathcal{T}$ to nodes of B, such that the following conditions are satisfied. For every branch of $\mathcal{T}$, $\mathcal{N}$ is one-to-one mapping between the nodes on this branch and the nodes of B, which is order preserving, in the

sense that, for all s, s' on this branch, if $\mathcal{N}(s)$ is an ancestor of $\mathcal{N}(s')$ in B, then s must be an ancestor of s' in $\mathcal{T}$. If $\mathcal{N}(s)$ is a node n with domain $\{v_1, \ldots, v_k\}$ and parents $p_1 \ldots, p_m$ in B, then the children of s in $\mathcal{T}$ are nodes $s_1, \ldots, s_k$, for which:

- $\mathcal{I}(s_i) = \mathcal{I}(s) \cup \{P_n(C_{v_i})\}$;
- $\mathcal{P}(s_i) = \mathcal{P}(s) \cdot \alpha$, where α is the entry in the table for n, that gives the conditional probability of $n = v_i$ given $p_1 = w_1, \ldots, p_m = w_m$, where each w_i is the unique value from the domain of p_i for which $P_{p_i}(C_{w_i}) \in \mathcal{I}(s)$.

It should be clear that every leaf s of such a B-process $\mathcal{T}$ describes an assignment of values to all nodes of B, i.e., every node n is assigned the unique value v for which $P_n(c_v) \in \mathcal{I}(s)$. Moreover, the probability $\mathcal{P}(s)$ of such a leaf is precisely the product of all the appropriate entries in the various conditional probability distributions. Therefore, the distribution $\pi_{\mathcal{T}}$ coincides with the distribution defined by the network of B.

We now construct a CP-theory CP_B, such that the execution models of CP_B will be precisely all B-processes. We first illustrate this process by showing how the Bayesian network in Figure 5.8 can be transformed into a CP-theory.

Example 5.8 (Sprinkler—cont'd). We can derive the following CP-theory from the Bayesian network in Figure 5.8.

$$\begin{aligned}
(Wet : 0.98) &\leftarrow \quad Sprinkler \wedge \quad Rain \\
(Wet : 0.8) &\leftarrow \quad Sprinkler \wedge \neg Rain. \\
(Wet : 0.9) &\leftarrow \neg Sprinkler \wedge \quad Rain. \\
(Wet : 0.0) &\leftarrow \neg Sprinkler \wedge \neg Rain. \\
(Sprinkler : 0.2). & \\
(Rain : 0.1). &
\end{aligned}$$

It should be obvious that the process in Figure 5.9 is an execution model of this theory and, therefore, that this theory defines the same probability distribution as the Bayesian network.

Again, this example exploits the fact that the random variables are all boolean, by using the more readable representation of Wet and $\neg Wet$ than that of $Wet(True)$ and $Wet(False)$.

It is now easy to see that the encoding used in the above example generalizes. Concretely, for every node n with parents $p_1, \ldots, p_m$ and domain $\{v_1, \ldots, v_k\}$, we should construct the set of all rules of the form:

$$(P_n(C_{v_1}) : \alpha_1) \vee \cdots \vee (P_n(C_{v_k}) : \alpha_k) \leftarrow P_{p_1}(C_{w_1}) \wedge \cdots P_{p_m}(C_{w_m}),$$

where each w_i belongs to the domain of p_i and each α_j is the entry for $n = v_j$, given $p_1 = w_1, \ldots, p_m = w_m$ in the CPT for n. Let us denote the CP-theory thus constructed by CP_B. The following result is then obvious.

Theorem 5.7. *Let B be a Bayesian network. Every B-process $\mathcal{T}$ is an execution model of the CP-theory* CP_B*, i.e., $\mathcal{T} \models \mathrm{CP}_B$. Therefore, the semantics of B coincides with the distribution π_C.*

This result shows that CP-logic offers a straightforward way of modelling the kind of processes described by a Bayesian network. We will now compare these two formalisms with respect to processes that do not directly fit into the Bayesian network structure. In the introduction, we already mentioned two reasons why this might happen: because multiple events are involved in determining the truth of a single property or because events propagate values in opposite directions.

5.6.2 Multiple causes for the same effect

In a process corresponding to a Bayesian network, the value of each random variable is determined by a single event. CP-logic, on the other hand, allows multiple events to affect the same property. This leads to better representations for effects that have a number of independent causes. Let us illustrate this by the following example.

Example 5.9. We consider a game of Russian roulette that is being played with two guns, one in the player's left hand and one in his right, each of which has a bullet in one of its six chambers.

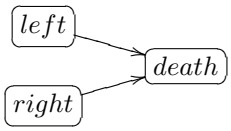
$$(Death : 1/6) \leftarrow Pull\_trigger(Left\_gun).$$
$$(Death : 1/6) \leftarrow Pull\_trigger(Right\_gun).$$

Figure 5.10 shows a Bayesian network for this example. The most obvious difference between these two representations concerns the independence between the two different causes for death. In the CP-theory, this independence is expressed by the *structure* of the theory, whereas in the Bayesian network, it is a *numerical* property of the probabilities in the conditional probability table for $Death$. Because of this, the CP-theory is more elaboration tolerant, since adding or removing an additional cause for $Death$ simply corresponds to adding or removing a single CP-event. Moreover, its representation is also more compact, requiring, in general, only n probabilities for n independent causes, instead of the 2^n entries that are needed in a Bayesian network table. Of course, these entries are nothing more than the result of applying a *noisy-or*[3] to the multiset of the probabilities with which each of the causes that are present actually causes the effect.

5.6.3 Cyclic causal relations

A second reason why a real life process might not correspond directly to a Bayesian network is because it may contain events that propagate values in opposite directions. We already saw this in Example 5.1, where angina could cause pneumonia, but, vice

[3]The *noisy-or* maps a multiset of probabilities α_i to $1 - \prod_i (1 - \alpha_i)$.



	left, right	left, ¬right	¬left, right	¬left, ¬right
death	11/36	1/6	1/6	0
¬death	25/36	5/6	5/6	1

Figure 5.10: A Bayesian network for Example .

versa, pneumonia could also cause angina. In CP-logic, such causal loops do not require any special treatment. For instance, the loop formed by the two CP-events

$$(Angina : 0.2) \leftarrow Pneumonia.$$
$$(Pneumonia : 0.3) \leftarrow Angina.$$

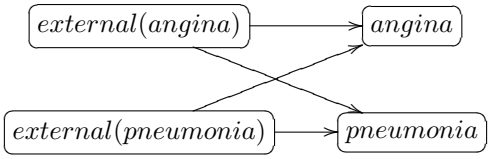
behaves as follows:

- If the patient has neither angina nor pneumonia by an external cause ('external' here does not mean exogenous, but simply that this cause is not part of the causal loop), then he will have neither;
- If the patient has angina by an external cause, then with probability 0.3 he will also have pneumonia;
- If the patient has pneumonia by an external cause, then with probability 0.2 he will also have angina;
- If the patient has both pneumonia and angina by an external cause, then he will obviously have both.

In order to get the same behaviour in a Bayesian network, this would have to be explicitly encoded. For instance, one could introduce new, artificial random variables $external(angina)$ and $external(pneumonia)$ to represent the possibility that $angina$ and $pneumonia$ result from an external cause and construct the Bayesian network that is shown in Figure 5.11. In general, to encode a causal loop formed by n properties, one would introduce n additional nodes, i.e., all of the n original properties would have the same n artificial nodes as parents.

5.7 CP-logic and logic programs

In the preceding sections, we identified the concept of a causal probabilistic event as a basic unit of causal information and defined a semantics for sets of such events in terms of constructive processes. There are some obvious similarities between these kind of processes and the kind of constructive processes that play a role in the various



	e(a),e(p)	e(a),¬e(p)	¬e(a),e(p)	¬e(a),¬e(p)
angina	1	1	0.2	0

	e(a),e(p)	e(a),¬e(p)	¬e(a),e(p)	¬e(a),¬e(p)
pneumonia	1	0.3	1	0

Figure 5.11: Bayesian network for the $angina$-$pneumonia$ causal loop.

kinds of fixpoint semantics we encountered in Chapter 2. Moreover, there is also an obvious similarity between the syntax of CP-logic and that of the rule sets we defined in Chapter 2. We now investigate these similarities. Concretely, we will first define a straightforward probabilistic extension of logic programs, called *Logic Programs with Annotated Disjunctions*, and then prove that this is essentially equivalent to CP-logic.

The connection between causal reasoning and logic programming has long been implicitly present; we can refer in this respect to, for instance, formalizations of situation calculus in logic programming (Pinto and Reiter 1993; Van Belleghem, Denecker, and De Schreye 1997). Here, we now make this relation explicit, by showing that the language of CP-logic, that we have constructed directly from causal principles, corresponds to existing logic programming concepts. In this respect, our work is similar to that of (McCain and Turner 1996), who defined the language of causal theories, which was then shown to be closely related to logic programming. However, as we will discuss later, McCain and Turner formalize somewhat different causal intuitions, which leads to a correspondence to a different logic programming semantics. Our results from this section will help to clarify the position of CP-logic among related work in the area of probabilistic logic programming, such as Poole's Independent Choice Logic (Poole 1997). Moreover, they provide additional insight into the role that causality plays in such probabilistic logic programming languages, as well as in normal and disjunctive logic programs.

5.7.1 Logic Programs with Annotated Disjunctions

In this section, we define the language of *Logic Programs with Annotated Disjunctions*, or *LPADs* for short. This is a probabilistic extension of logic programming, which is based on disjunctive logic programs. This is a natural choice, because disjunctions themselves—and therefore also disjunctive logic programs—already represent a kind of uncertainty. Indeed, to give just one example, we could use these to model indeterminate effects of actions. Consider, for instance, the following disjunctive rule:

$$Heads \vee Tails \leftarrow Toss.$$

This offers a quite intuitive representation of the fact that tossing a coin will result in either heads or tails. Of course, this is not all we know. Indeed, a coin also has equal probability of landing on heads or tails. The idea behind LPADs is now simply to express this by annotating each of the disjuncts in the head with a probability, i.e., we write:

$$(Heads : 0.5) \vee (Tail : 0.5) \leftarrow Toss.$$

Formally, an LPAD is a set of rules:

$$(h_1 : \alpha_1) \vee \cdots \vee (h_n : \alpha_n) \leftarrow \varphi, \tag{5.28}$$

where the h_i are ground atoms and φ is a sentence. As such, LPADs are syntactically identical to CP-logic. However, we define their semantics quite differently. For instance, the above example expresses that precisely one of the following logic programming rules holds: either $Heads \leftarrow Toss$ holds, i.e., if the coin is tossed this will yield heads, or the rule $Tails \leftarrow Toss$ holds, i.e., tossing the coin gives tails. Each of these two rules has a probability of 0.5 of being the actual instantiation of the disjunctive rule.

More generally, every rule of form (5.28) represents a probability distribution over the following set of logic programming rules:

$$\{(h_i \leftarrow \varphi) \mid 1 \leq i \leq n\}.$$

From these distributions, a probability distribution over logic programs is then derived. To formally define this distribution, we introduce the following concept of a *selection*. We use the notation $head^*(r)$ to denote the set of pairs $head(r) \cup \{(\emptyset, 1 - \sum_{(h:\alpha)\in head(r)} \alpha)\}$, where $\emptyset$ represents the possibility that none of the h_i's are caused by the rule r.

Definition 5.14 (C-selection). Let C be an LPAD. A *C-selection* is a function σ from C to $\bigcup_{r\in C} head^*(r)$, such that for all $r \in C$, $\sigma(r) \in head^*(r)$. By $\sigma^h(r)$ and $\sigma^\alpha(r)$ we denote, respectively, the first and second element of the pair $\sigma(r)$. The set of all C-selections is denoted as $\mathcal{S}_C$.

The probability $P(\sigma)$ of a selection σ is now defined as $\prod_{r\in C} \sigma^\alpha(r)$. For a set $S \subseteq \mathcal{S}_C$ of selections, we define the probability $P(S)$ as $\sum_{\sigma\in S} P(\sigma)$. By C^σ we denote the logic program that consists of all rules $\sigma^h(r) \leftarrow body(r)$ for which $r \in C$ and $\sigma^h(r) \neq \emptyset$. Such a C^σ is called an *instance* of C. We will interpret these instances by the well-founded model semantics. Recall that, in general, the well-founded model $wfm(P)$ of a program P is a pair (I, J) of interpretations, where I contains all atoms that are certainly true and J contains all atoms that might possibly be true. If $I = J$, then the well-founded model is called exact. Intuitively, if $wfm(P)$ is exact, then the truth of all atoms can be decided, i.e., everything that is not false can be derived. In the semantics of LPADs, we wanted to ensure that all uncertainty is expressed by means of the annotated disjunctions. In other words, given a specific selection, there should no longer be any uncertainty. We therefore impose the following criterion.

Definition 5.15 (Soundness). An LPAD C is *sound* iff all instances of C have an exact well-founded model.

For such LPADs, the following semantics can now be defined.

Definition 5.16 (Instance based semantics μ_C). Let C be a sound LPAD. For an interpretation I, we denote by $W(I)$ the set of all C-selections σ for which $wfm(C^\sigma) = (I, I)$. The *instance based semantics* μ_C of C is the probability distribution on interpretations, that assigns to each I the probability $P(W(I))$ of this set of selections $W(I)$.

It is straightforward to extend this definition to allow for exogenous predicates as well. Indeed, in Section 2.2, we have already seen how to define the well-founded semantics for rule sets with open predicates, and this is basically all that is needed. Concretely, given an interpretation X for a set of exogenous predicates, we can define the instance based semantics μ_C^X given X as the distribution that assigns, to each interpretation I of the endogenous predicates, the probability of the set of all selections σ for which (I, I) is the well-founded models of C^σ given (X, X) (as defined in Section 2.2). Of course, this semantics is only defined for LPADs that are sound in X, meaning that the well-founded model of each C^σ given (X, X) is two-valued.

5.7.2 Equivalence to CP-logic

Every CP-theory is syntactically also an LPAD and vice versa. The key result of this section is now that the instance based semantics μ_C for LPADs coincides with the CP-logic semantics π_C defined in Section 5.2.

Theorem 5.8. *Let C be a CP-theory that is valid in X. Then C is also an LPAD that is sound in X and, moreover, $\mu_C^X = \pi_C^X$.*

We remark that it is not the case that every sound LPAD is also a valid CP-theory. In other words, there are some sound LPADs that cannot be seen as a sensible description of a set of causal probabilistic events.

Example 5.10. It is easy to see that the following CP-theory has no execution models.

$$\begin{aligned}(P : 0.5) \vee (Q : 0.5) &\leftarrow R.\\ R &\leftarrow \neg P.\\ R &\leftarrow \neg Q.\end{aligned}$$

However, each of its instances has an exact well-founded model: for $\{P \leftarrow R; R \leftarrow \neg P; R \leftarrow \neg Q\}$ this is $\{R, P\}$ and for $\{Q \leftarrow R; R \leftarrow \neg P; R \leftarrow \neg Q\}$ this is $\{R, Q\}$. It does not seem possible to interpret this CP-theory in a reasonable way as an enumeration of all the relevant CP-events in some domain.

5.7.3 Discussion

The results of this section relate CP-logic to LPADs and, more generally speaking, to the area of logic programming and its probabilistic extensions. As such, these results

help to position CP-logic among related work, such as Poole's Independent Choice Logic and McCain and Turner's causal theories, which we will discuss in Section 5.9.1. Moreover, they can also be seen as providing a valuable piece of knowledge representation methodology for these languages, as they clarify how causal information can be modeled in these languages. To illustrate, we now discuss the relevance of our theorem for some logic programming variants.

Disjunctive logic programs. In probabilistic modelling, it is often useful to consider the qualitative structure of a theory separately from its probabilistic parameters. Indeed, for instance, in machine learning, the problems of structure learning and parameter learning are two very different tasks. If we consider only the structure of a CP-theory, then, syntactically speaking, we end up with a *disjunctive logic program*, i.e., a set of rules:

$$h_1 \vee \cdots \vee h_n \leftarrow \varphi. \tag{5.29}$$

We can also single out the qualitative information contained in the semantics π_C of such a CP-theory. Indeed, as we have already seen, like any probability distribution over interpretations, π_C induces a possible world semantics, consisting of those interpretations I for which $\pi_C(I) > 0$. Now, let us restrict our attention to only those CP-theories in which, for every CP-event r, the sum of the probabilities α_i appearing $head(r)$ is precisely 1 and, moreover, every such $\alpha_i > 0$. It is easy to see that this set of possible worlds is then independent of the precise values of the α_i, i.e., the qualitative aspects of the semantics of such a theory depend only on the qualitative aspects of its syntactical form.

From the point of view of disjunctive logic programming, this set of possible worlds therefore offers an alternative semantics for such a program. Under this semantics, the intuitive reading of a rule of form (5.29) is: "φ causes a non-deterministic event, whose effect is precisely one of h_1,..., h_n." Clearly, this is a different informal reading than in the standard stable model semantics for disjunctive programs (Przymusinski 1991). Indeed, under our reading, a rule corresponds to a causal event, whereas, under the stable model reading, it is supposed to describe an aspect of the reasoning behaviour of a rational agent. Consider, for instance, the disjunctive program $\{p \vee q.\ p.\}$. To us, this program describes a set of two non-deterministic events: One event causes either p or q and another event always causes p. Formally, this leads to two possible worlds, namely $\{p\}$ and $\{p, q\}$. Under the stable model semantics, however, this program states that an agent believes either p or q and the agents believes p. In this case, he has no reason to believe q and the only stable model is $\{p\}$. So, clearly, CP-logic treats disjunction in a fundamentally different way than the stable semantics. Interestingly, the *possible model semantics* (Sakama and Inoue 1994) for disjunctive programs is quite similar to our treatment. Indeed, it consists of the stable models of instances of a program. Because, as shown in Section 5.7.2, the semantics of CP-logic considers the well-founded models of instances, these two semantics are very closely related. Indeed, for a large class of programs, including all stratified ones, they coincide completely.

Normal logic programs. Let us consider a logic program P, consisting of a set of rules $h \leftarrow \varphi$, with h a ground atom and φ a formula. Syntactically, such a program

is also a deterministic CP-theory. Its semantics π_P assigns a probability of 1 to a single interpretation and 0 to all other interpretations. Moreover, the results from Section 5.7.2 tell us that the interpretation with probability 1 will be precisely the well-founded model of P. As such, these results show that a logic program under the well-founded semantics can be viewed as a description of deterministic causal information. Concretely, we find that we can read a rule $h \leftarrow \varphi$ as: "φ causes a deterministic event, whose effect is h."

This observation makes explicit the connection between causal reasoning and logic programming that has long been implicitly present in the field of logic programming, as is witnessed, e.g., by the work on situation calculus in logic programming. As such, it enhances the theoretical foundations behind the pragmatic use of logic programs to represent causal events.

ID-logic. Because an inductive definition in ID-logic is represented by a logic program under the well-founded semantics, our results show that finite definitions in ID-logic are, both syntactically and semantically, identical to deterministic CP-theories. It is interesting that an attempt to formalize the well-known mathematical principle of definition by induction should yield the same formal language as our attempt to formalize causal intuitions. One plausible explanation for this phenomenon is that perhaps the appeal that inductive definitions hold for a mathematician stems from experience with the causal laws that govern the behaviour of physical systems. Or to put it another way, perhaps an inductive definition is nothing more than a description of a causal process, that takes place not in the real world, but in the domain of mathematical objects.

5.8 Origins of CP-logic

The work on CP-logic grew actually out of research on probabilistic logic programming and, in particular, the development of logic programs with annotated disjunctions. In trying to clarify the intuitive meaning of expressions in this language, we often found it useful—and sometimes even necessary—to refer to causality. For instance, the ability to have multiple rules with the same atom in the head turned out to be quite convenient, as illustrated by our formalization of the Russian roulette example:

$$(Death : 1/6) \leftarrow Fire(Left\_gun).$$
$$(Death : 1/6) \leftarrow Fire(Right\_gun).$$

This correctly represents the example and, moreover, it does so in a concise and appealing way. However, it is clearly not possible to say that, for instance, the first rule should be read as: "If the left gun is fired, then the probability of death is 1/6." (If the probability of firing the right gun is non-zero, the statement is obviously false.) Examining the intuitions that had lead to this theory, we found that we actually thought of this rule as representing only one *possible cause* for death. So, we started to interpret the '$\leftarrow$'-connective of LPADs as a statement about causes and effects.

However, it was not clear that this interpretation was actually sanctioned in any way by the formal semantics that we had constructed for this language, because this just

considered the well-founded semantics of certain probabilistically constructed logic programs and, as such, did not seem to contain any features particularly related to causality. We therefore decided to construct a different semantics—one that did explicitly formalize causal intuitions—to see how this would compare to our original semantics.

Gradually, it became apparent that by constructing a causal interpretation for our logic programming style constructs, we were actually also learning something about the nature of causality itself. Namely, it became clear that, to a much larger extent than common in the literature, our logic was focusing on a dynamic aspect of causality. Indeed, the concept of an event, something which actually *happens* at some point in time, became central to not only our formal semantics, but also to our conception of causality itself. We therefore realized that, instead of just defining an alternative, causal semantics for some given probabilistic logic programming language, our work was actually examining something that was inherently present in causality itself. This lead to a final shift in focus, where we now constructed a logic from the ground up—both in form and in meaning—based only on intuitions regarding causality. Our original semantics for logic programs with annotated disjunctions still proved useful, however, because it allowed us to relate our new causal language of CP-logic to existing logic programming concepts, as we have seen in this section. In this way, we showed that a causal interpretation of the '$\leftarrow$'-connective not only makes sense in the context CP-logic, but also for logic programs.

5.9 Related work

In this chapter, we discuss some research that is related to our work on CP-logic. Roughly speaking, we can divide this into two different categories, namely, the related work that focuses mainly on formalizing causality and that which focuses mainly on representing probabilistic knowledge.

5.9.1 Causal languages

Our work on CP-logic is primarily intended as a study of the dynamic nature of causal information from a knowledge representation perspective. As such, it is closely related to the work of Pearl. In Section 5.6, we have already compared CP-logic to Pearl's formal tool of Bayesian networks and showed that it offers certain representational advantages through its more flexible and fine-grained representation of causal events. We have, however, not yet discussed the fact that the focus of our study is somewhat different from Pearl's. Indeed, his work focuses on the behaviour of causal models in the presence of *interventions*, i.e., outside manipulations that preempt the normal behaviour of the system. This is a topic that is somewhat orthogonal to our work. Indeed, while Pearl examines interventions in the formal context of structural models (a generalization of Bayesian networks), it seems equally possible to do this in the setting of CP-logic. In fact, this is actually a promising direction for future research. Indeed, in CP-logic, one could consider interventions that preempt, add or replace just

a single CP-event. This allows more fine-grained manipulations of a causal model than are possible using Bayesian networks or structural models.

Moreover, one of the interesting uses of interventions is the handling of counterfactuals, which have been used by Halpern to define concepts such as "actual causes" (Halpern and Pearl 2001a) and "explanations" (Halpern and Pearl 2001b). The explicitly dynamic processes of CP-logic offer an interesting setting in which to investigate these concepts as well. Indeed, in any particular branch of an execution model of a CP-theory every true atom p is caused by at least one CP-event, whose precondition φ was satisfied at the time when this event happened. It now seems sensible to call φ an actual cause of p. An interesting question is to what extent such a definition would coincide with the notion of actual causation defined by Halpern.

Another attempt to formalize causal knowledge, which, like CP-logic, also has a close relation to logic programming, are McCain and Turner's *causal theories* (McCain and Turner 1996). A causal theory is a set of rules $\varphi \Leftarrow \psi$, where φ and ψ are propositional formulas. The semantics of such a theory T is defined by a fixpoint criterion. Concretely, an interpretation I is a model of T if I is the *unique* classical model of the theory T^I that consists of all φ, for which there is a rule $\varphi \Leftarrow \psi$ in T such that $I \models \psi$.

In CP-logic, we assume that the domain is initially in a certain state, which then changes through series of events. This naturally leads to the kind of constructive processes that we have used to define the formal semantics of CP-logic. By contrast, according to McCain and Turner's fixpoint condition, a proposition can have any truth value, as long as their exists some causal explanation for this truth value. This difference mainly manifests itself in two ways.

First, in CP-logic, every endogenous property has an initial truth value, which can only change as the result of an event. As such, there is a fundamental asymmetry between falsity and truth, since only one of them represents the "natural" state of the property. For McCain & Turner, however, truth and falsity are completely symmetric and both need to be causally explained. As such, if the theory is to have any models, then, for every proposition Q, there must always be a cause for either Q or $\neg Q$.

A second difference is that the constructive processes of CP-logic rule out any unfounded causality, i.e., it cannot be the case that properties spontaneously cause themselves. In McCain & Turner's theories, this is allowed to happen. For instance, the CP-theory $\{Q \leftarrow Q\}$ has $\{\}$ as its model, whereas the causal theory $\{Q \Leftarrow Q\}$ has $\{Q\}$ as its model. As such, the direct representation of cyclic causal relations that is possible in CP-logic (e.g., Example 5.1) cannot be done in causal theories; instead, one has to use an encoding similar to the one needed in Bayesian networks (e.g., Figure 5.11). In practice, the main advantage of McCain & Turner's treatment of causal cycles seems to be that it offers a way of introducing exogenous atoms into the language. Indeed, by including both $Q \Leftarrow Q$ and $\neg Q \Leftarrow \neg Q$, one can express that Q can have any truth value, without this requiring any further causal explanation. Of course, CP-logic has no need for such a mechanism, since we make an explicit distinction between exogenous and endogenous predicates. It is interesting to observe that, given the relation between logic programming and causal theories proven in (McCain 1997), this difference can actually be traced back to the difference between the well-founded and completion semantics for logic programs.

5.9.2 Probabilistic languages

In this section, we compare CP-logic to a number of probabilistic logic programming languages. Let us first and foremost point out that, in general, our main contribution to this field does not consist of some original new formal language, but, rather, of a new intuitive understanding of the "same old" mathematical objects. Indeed, whereas probabilistic logic programming research typically takes some logic programming variant as its starting point, we have begun from fundamental observations about the nature of causality, without assuming any *a priori* relation to logic programs. Nevertheless, our attempt to formalize intuitions about causal probabilistic events has lead to a formal language that is remarkably close to probabilistic logic programming. On the one hand, this demonstrates that probabilistic logic programming can be used to deal with causality. On the other hand, it also allows such languages to be understood, motivated and explained from the ground up, based only on the concept of causal probabilistic events, without any reference to prior logic programming developments.

We will now discuss a number of probabilistic logic programming languages in some more detail.

Independent Choice Logic

Independent Choice Logic (ICL) (Poole 1997) by Poole is a probabilistic extension of abductive logic programming, that extends the earlier formalism of *Probabilistic Horn Abduction* (Poole 1993). An ICL theory consists of both a logical and a probabilistic part. The logical part is an acyclic logic program. The probabilistic part consists of a set of rules of the form (in CP-logic syntax):

$$(h_1 : \alpha_1) \vee \cdots \vee (h_n : \alpha_n).$$

The atoms h_i in such clauses are called *abducibles*. Each abducible may only appear once in the probabilistic part of an ICL program; in the logical part of the program, abducibles may only appear in the bodies of clauses.

Syntactically speaking, each ICL theory is also CP-theory. Moreover, the ICL semantics of such a theory (as formulated in, e.g., (Poole 1997)) can easily be seen to coincide with our instance based semantics for LPADs. As such, an ICL theory can be seen as a CP-theory in which every CP-event is either deterministic or unconditional.

We can also translate certain LPADs to ICL in a straightforward way. Concretely, this can be done for acyclic LPADs without exogenous predicates, for which the bodies of all CP-events are conjunctions of literals. Such a CP-event r of the form:

$$(h_1 : \alpha_1) \vee \cdots \vee (h_n : \alpha_n) \leftarrow \varphi$$

is then transformed into the set of rules:

$$\left\{ \begin{array}{r} h_1 \leftarrow \varphi \wedge Choice_r(1). \\ \cdots \\ h_n \leftarrow \varphi \wedge Choice_r(n). \\ (Choice_r(1) : \alpha_1) \vee \cdots \vee (Choice_r(n) : \alpha_n). \end{array} \right\}$$

The idea behind this transformation is that every selection of the original theory C corresponds to precisely one selection of the translation C'. More precisely, if we denote by $ChoiceRule(r)$ the last CP-event in the above translation of a rule r, then a C-selection σ corresponds to the C'-selection σ', for which for all $r \in C$, $\sigma(r) = (h_i, \alpha_i)$ iff $\sigma'(ChoiceRule(r)) = (Choice_r(i), \alpha_i)$. It is quite obvious that this one-to-one correspondence preserves both the probabilities of selections and the (restrictions to the original alphabet of the) well-founded models of the instances of selections. This suffices to show that the probability distribution defined by C coincides with the (restriction to the original alphabet of) the probability distribution defined by C'.

So, our result on the equivalence between LPADs and CP-logic shows that the two parts of an ICL theory can be understood as, respectively, a set of unconditional probabilistic events and a set of deterministic causal events. In this sense, our work offers a causal interpretation for ICL. It is, in this respect, somewhat related to the work of Finzi et al. on causality in ICL. In (Finzi and Lukasiewicz 2003), these authors present a mapping of ICL into Pearl's structural models and use this to derive a concept of actual causation for this logic, based on the work by Halpern (Halpern and Pearl 2001a). This approach is, however, somewhat opposite to ours. Indeed, we view the event-based structure of CP-logic as a more fine-grained model of causality. Transforming a CP-theory into a structural model actually loses information, in the sense that it is not possible to recover the original structure of the theory. From the point-of-view of CP-logic, the approach of Finzi et al. would therefore not make much sense, since it would attempt to define the concept of actual causation in a more fine-grained model of causal information by means of a transition to a coarser one.

P-log

P-log (Baral, Gelfond, and Rushton 2004) is an extension of the language of Answer Set Prolog with new constructs for representing probabilistic information. It is a sorted logic, which allows for the definition of *attributes*, which map tuples (of particular sorts) into a value (of a particular sort). Two kinds of probabilistic statements are considered. The first are called *random selection rules* and are of the form:

$$[r]\ random(A(\mathbf{t}) : \{x : P(x)\}) \leftarrow \varphi.$$

Here, r is a name for the rule, P is an unary boolean attribute, A is an attribute with $\mathbf{t}$ a vector of arguments of appropriate sorts, and φ is a collection of so-called extended literals[4]. The meaning of a statement of the above form is that, if the body φ of the rule is satisfied, the attribute $A(\mathbf{t})$ will take on a value from the intersection of its domain with the set of all terms x for which $P(x)$ holds. The choice of which value will be assigned to this attribute is random and, by default, all possible values are considered equally likely. It is, however, possible to override such a default, using the second kind of statements, called *probabilistic atoms*. These are of the form:

$$pr_r(A(\mathbf{t}) = y \mid_c \varphi) = \alpha.$$

[4] An extended literal is either a classical literal or a classical literal preceded by the default negation not, where a classical literal is either an atom $A(\mathbf{t}) = t_0$ or the classical negation $\neg A(\mathbf{t}) = t_0$ thereof.

Such a statement should be read as: If φ holds, then the probability of attribute $A(\mathbf{t})$ taking on value y due to the random selection process described by rule r is α.

The information expressed by a random selection rule and it associated probabilistic atoms is somewhat similar to a CP-event, but stays closer to a Bayesian network style representation. Indeed, it expresses that, under certain conditions, the value of a certain attribute will be determined by some implicit random process, which produces each of a number of possible outcomes with a certain probability. We see that, as in Bayesian networks, there is no way of directly representing information about the actual events that might take place; instead, only information about the way in which they eventually affect the value of some attribute (or random variable, in Bayesian network terminology) can be incorporated. Therefore, representing the kind of phenomena discussed in Section 5.6—namely, cyclic causal relations and effects with a number of independent possible causes—requires the same kind of encoding in P-log as in Bayesian networks.

A second interesting difference is that a $random$-statements of P-log represent an experiment in which a value is selected from a *dynamic* set of alternatives, whereas, in CP-logic the set of possible outcomes is specified statically. Consider, for instance, a robot that leaves a room by selecting at random one of the doors that happens to be open. In P-log, this can easily be written down as:

$$[r]\ random(Leave\_through : \{x : Open\_door(x)\}).$$

In CP-logic, such a concise representation is currently not possible.

Apart from probabilistic statements, a P-log program can also contain a set of regular Answer Set Prolog rules and a set of observations and actions. The difference between observations and actions is the same as highlighted by (Pearl 2000), i.e., observations are supposed to have been generated by the causal processes described by the theory, whereas actions explicitly interfere with the normal state of affairs and, therefore, cannot and should not be explained by the theory. As such, the scope of P-log is significantly broader than that of CP-logic and it is clearly a more full-blown knowledge representation language than CP-logic, which is only aimed at expressing a specific kind of causal knowledge.

First-order Versions of Bayesian networks

In this section, we discuss two approaches that aim at lifting the propositional formalism of Bayesian networks to a first-order representation, namely *Bayesian Logic Programs (BLPs)* (Kersting and Raedt 2000) and *Relational Bayesian Networks (RBNs)* (Jaeger 1997).

A Bayesian Logic Program or BLP consists of a set of definite clauses, using the symbol "|" instead of "←", i.e., clauses of the form

$$P(\mathbf{t}_0) \mid B_1(\mathbf{t_1}), \ldots, B_n(\mathbf{t_n}).$$

in which P and the B_i's are predicate symbols and the $\mathbf{t_j}$'s are tuples of terms. For every predicate symbol P, there is a domain $dom(P)$ of possible values. The meaning of such a program is given by a Bayesian network, whose nodes consist of all the atoms

in the least Herbrand model of the program. The domain of a node for a ground atom $P(\mathbf{t})$ is $dom(P)$. For every ground instantiation $P(\mathbf{t_0}) \mid B_1(\mathbf{t_1}), \ldots, B_n(\mathbf{t_n})$ of a clause in the program, the network contains an edge from each $B_i(\mathbf{t_i})$ to $P(\mathbf{t_0})$, and these are the only edges that exist.

To complete the definition of this Bayesian network, all the relevant conditional probabilities also need to be defined. To this end, the user needs to specify, for each clause in the program, a conditional probability table, which defines the conditional probability of every value in $dom(P)$, given an assignment of values to the atoms in the body of the clause. Now, let us first assume that every ground atom in the Bayesian network is an instantiation of the head of precisely one clause in the program. In this case, the tables for the clauses suffice to determine the conditional probability tables of the network, because every node can then simply take its probability table from this unique clause. However, in general, there might be many such clauses. To also handle this case, the user needs to specify, for each predicate symbol P, a so-called *combination rule*, which is a function that produces a single probability from a multiset of probabilities. The conditional probability table for a ground atom $P(\mathbf{t})$ can then be constructed from the set of all clauses r, such that $P(\mathbf{t})$ is an instantiation of $head(r)$, by finding the appropriate entries in the tables for all such clauses r and then applying the combination rule for P to the multiset of these values. According to the semantics of Bayesian Logic Programs, this combination rule will always be applied, even when there exists only a single such r.

This completes the definition of BLPs as given in, e.g., (Kersting and Raedt 2000). More recently, a number of issues with this formalism have lead to the development of Logical Bayesian Networks (Fierens, Blockeel, Bruynooghe, and Ramon 2005). These issues have also prompted the addition of so-called "logical atoms" to the original BLP language (Kersting and Raedt 2007). Since this does not significantly affect any of the comparisons made in this section, however, we will ignore this extension.

A *Relational Bayesian Network* is a Bayesian network in which the nodes correspond to predicate symbols and the domain of a node for a predicate P/n consists of all possible interpretations of this predicate symbol in some fixed domain D, i.e., all subsets of D^n. The conditional probability distribution associated to such a node P is specified by a *probability formula* F_p. For every tuple $\mathbf{d} \in D^n$, $F_p(\mathbf{d})$ defines the probability of $\mathbf{d}$ belonging to the interpretation of P in terms of probabilities of tuples $\mathbf{d}'$ belonging to the interpretation of a predicate P', where P' is either a parent of P in the graph or even, under certain conditions, P itself. Such a probability formula can contain a number of different operations on probabilities, including the application of arbitrary combination rules. Such a Relation Bayesian Network can also be compiled into a network that is similar to that generated by a BLP, i.e., one in which the nodes correspond to domain atoms instead of predicate symbols. The main advantage of such a compiled network is that it allows more efficient inference.

Again, the main difference between these two formalisms and CP-logic is that they both stick to the Bayesian network style of modelling, in the sense that the actual processes and events that determine the values of the random variables are entirely abstracted away and only the resulting conditional probabilities are retained. However, through the use of, respectively, combination rules and probability formulas, these can be represented in a more structured manner than in a simple table. In this way, knowl-

edge about, for instance, the underlying causal events can be exploited to represent the conditional probability distributions in a concise way. The most common example is probably the use of the *noisy-or* to handle an effect which has a number of independent possible causes. For instance, let us consider the Russian roulette problem of Example 5.9. In a BLP, the relation between the guns firing and the player's death could be represented by the following clause:

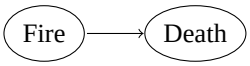
$$Death \mid Fire(X).$$

	$Fire(x) = \mathbf{t}$	$Fire(x) = \mathbf{f}$
$Death = \mathbf{t}$	$1/6$	0
$Death = \mathbf{f}$	$5/6$	1

Combination rule for $Death$: *noisy-or*

In Relational Bayesian Networks, this would be represented as follows:

$$F_{Death} = \textit{noisy-or}(\{1/6 \cdot Fire(x) \mid x\})$$



As such, combination rules do allow some knowledge about the events underlying the conditional probabilities to be incorporated into the model. However, this is of course not the same as actually having a structured representation of the events themselves, as is offered by CP-logic. As a consequence of this, cyclic causal relations, such as that of our $Pneumonia$-$Angina$ example, still need the same kind of encoding as in a Bayesian network.

Other approaches

In this section, we give a quick overview of some other related languages. An important class of probabilistic logic programming formalisms are those following the *Knowledge Based Model Construction* approach. Such formalisms allow the representation of an entire "class" of propositional models, from which, for a specific query, an appropriate model can then be constructed "at run-time". This approach was initiated by Breese (Breese 1992) and Bacchus (Bacchus 1993) and is followed by both Bayesian Logic Programs and Relational Bayesian Networks. Other formalism in this class are *Probabilistic Knowledge Bases* of Ngo and Haddawy (Ngo and Haddawy 1997) and *Probabilistic Relational Models* of Getoor et al. (Getoor, Friedman, Koller, and Pfeffer 2001). From the point of view of comparison to CP-logic, both are very similar to Bayesian Logic Programs (see, e.g., (Kersting and Raedt 2001) for a comparison).

The language used in the *Programming in Statistical Modelling* system (PRISM) (Sato and Kameya 1997) is very similar to Independent Choice Logic. Our comments concerning the relation between CP-logic and Independent Choice Logic therefore carry over to PRISM.

Like CP-logic, *Many-Valued Disjunctive Logic Programs* (Lukasiewicz 2001) are also related to disjunctive logic programming. However, in this language, probabilities

are associated with disjunctive clauses as a whole. In this way, uncertainty of the implication itself—and not, as is the case with LPADs or CP-logic, of the disjuncts in the head—is expressed.

All the works mentioned so far use point probabilities. There are however also a number of formalisms using probability intervals: *Probabilistic Logic Programs* of Ng and Subrahmanian (Ng and Subrahmanian 1992), their extension to *Hybrid Probabilistic Programs* of Dekhtyar and Subrahmanian (Dekhtyar and Subrahmanian 2000) and *Probabilistic Deductive Databases* of Lakshmanan and Sadri (Lakshmanan and Sadri 1994). Contrary to our approach, programs in these formalisms do not define a $single$ probability distribution, but rather a *set* of possible probability distributions, which allows one to express a kind of "meta-uncertainty", i.e., uncertainty about which probability distribution is the "right" one. Moreover, the techniques used by these formalisms tend to have more in common with constraint logic programming than standard logic programming. The more recent formalism of CLP(BN) (Costa, Page, Qazi, and Cussens 2003) belongs to this class.

We also want to mention *Stochastic Logic Programs* of Muggleton and Cussens (Cussens 2000; Muggleton 2000), which is a probabilistic extension of Prolog. In this formalism, probabilities are attached to the selection of clauses in Prolog's SLD-resolution algorithm, which basically results in a first-order version of stochastic context free grammars. Because of this formalism's strong ties to the procedural aspects of Prolog, it appears to be quite different from CP-logic and indeed all of the other formalisms mentioned here.

ProbLog (De Raedt, Kimmig, and Toivonen 2007) is a more recent probabilistic extension of pure Prolog. Here, too, every clause is labeled with a probability. The semantics of ProbLog is very similar to that of LPADs and, in fact, the semantics of a *ground* ProbLog program coincides completely with that of the corresponding LPAD. More precisely put, a ProbLog rule of the form:

$$\alpha: \quad h \leftarrow b_1, \ldots, b_n,$$

where h and the b_i are ground atoms is entirely equivalent to the LPAD rule:

$$(h:\alpha) \leftarrow b_1, \ldots, b_n.$$

For non-ground programs, however, there is a difference. The semantics of an LPAD first grounds the entire program and then probabilistically selects instantiations of the rules of this ground program. In ProbLog, on the other hand, selections directly pick out rules of the original program. This means that, for instance, the following ProbLog-rule:

$$0.8: \quad likes(X,Y) \leftarrow likes(X,Z), likes(Z,Y),$$

specifies that, with probability 0.8, the $likes$-relation is entirely transitive, whereas the corresponding LPAD-rule would mean that for all *individuals* a, b and c, the fact that a likes b and b likes c causes a to like c with probability 0.8.

5.10 Conclusions and future work

The work presented in this chapter is primarily intended as a study of the nature of causality. We have argued that causal statements have an inherent dynamic aspect, which leads naturally to the notion of a *causal probabilistic event description* as an important unit of causal knowledge. The intuitive meaning of such statements strongly suggests a formal semantics that is defined in terms of constructive probabilistic processes. We have worked out this idea in the language of CP-logic. An important theorem regarding this logic is that all execution models of a CP-theory generate the same probability distribution over their final states. This shows that it is possible to view CP-logic as a probabilistic modelling language, that refines Bayesian networks.

Our main contribution is that we have shown the syntax and semantics of our language to follow naturally from an initial analysis of causal statements. Of course, certain specific choices could be argued with—for instance, our representation of the possible effects of an event is perhaps somewhat oversimplified and the immediate intuitive appeal of our semantics is clearly less obvious for theories containing negation than for positive theories. However, the basic structure of CP-logic and its semantics are essentially already implied by the concept of a causal probabilistic event description itself.

We have also shown that the formal semantics of CP-logic is closely related to logic programming. Because of the way in which we have constructed our logic as a natural formalization of causal statements, this shows that logic programming constructs can be interpreted in a causal way. To be more concrete, we have shown that a normal logic program under the well-founded semantics can be understood as a set of deterministic causal statements, we have presented an alternative semantics for disjunctive logic programs (similar to that of (Sakama and Inoue 1994)) under which these can be interpreted as sets of non-deterministic causal events, and we have shown that a theory in Poole's independent choice logic (Poole 1997) can be understood as a combination of deterministic causal events and unconditional probabilistic events.

The fact that CP-logic can be seen as a refinement of Bayesian networks, and is as expressive as independent choice logic, suggests that it might be well suited for modelling practical applications. Investigating this further would be especially interesting, since it would also shed light on how the kind of causal knowledge we have been studying actually appears in real application domains. To make CP-logic more suitable for practical purposes, it could still be improved in a number of ways. To be more concrete, we see the following opportunities for future research.

Refinement of CP-logic. The current language of CP-logic is restricted in a number of ways. First, it only allows a finite number of causal probabilistic events. Let us consider, for instance, a die that is rolled as long as it takes to obtain a six. Here, there is no upper bound on the number of throws that might be needed and, therefore, this example can currently not be represented in CP-logic. Second, CP-logic is also limited in its representation of the effects of an event. For instance, it is not possible to represent events whose range of possible outcomes is not completely fixed beforehand. Third, CP-logic currently can only handle properties that are either fully present or fully

absent. As such, it cannot correctly represent causes which have only a contributory effect, e.g., turning on a tap would not instantaneously cause a basin to be full, but only contribute a certain amount per time unit.

Integration into a larger formalism. To correctly formalize a domain in CP-logic, a user must exactly know the causes and effects of all relevant events that might happen. For real domains of any significant size, this is an unrealistic assumption. Indeed, typically, one will only have such detailed knowledge about certain parts of a domain. So, in order to still be able to use CP-logic in such a setting, it would have to be integrated with other forms of knowledge. There are some obvious candidates for this: statements about the probabilities of certain properties, statements about probabilistic independencies (such as those in Bayesian networks), and constraints on the possible states of the domain. Integrating these different forms of knowledge without losing conceptual clarity is one of the main challenges for future work regarding CP-logic, and perhaps even for the area of uncertainty in artificial intelligence as a whole.

Inference. The most obvious inference task in the context of CP-logic is calculating the probability $\pi_C(\varphi)$ of a formula φ. A straightforward way of doing this would be to exploit the relation between CP-logic and (probabilistic) logic programming, such that we perform these computations by reusing existing algorithms (e.g., the inference algorithm of Poole's independent choice logic (Poole 1997)) in an appropriate way. A more advanced technique, using binary decision diagrams, is currently being studied by Riguzzi [personal communication]. Another interesting inference task concerns the construction of a theory in CP-logic. For probabilistic modelling languages in general, it is typically not desirable that a user is forced to estimate or compute concrete probability values herself; instead, it should be possible to automatically derive these from a given data set. For CP-logic, there already exist algorithms that are able to do this in certain restricted cases (Riguzzi 2004; Blockeel and Meert 2007). It would be interesting to generalize these, in order to make them generally applicable. Besides such learning of probabilistic parameters, it is also possible to learn the structure of the theory itself. This too is an important topic, because if we are able to construct the theory that best describes a given data set, we are in effect finding out which causal mechanisms are most present in this data. Such information can be relevant for many domains. For instance, when bio-informatics attempts to distinguish active from non-active compounds, this is exactly the kind of information that is needed.

Chapter 6

Proofs of the theorems

In this chapter, we present proofs of the theorems that were stated in the previous chapter. To ease notation, we will assume that there are no exogenous predicates. This can be done without loss of generality, since all our results can simply be relativized with respect to some fixed interpretation for these predicates.

6.1 Semantics is well-defined

We start by proving that the semantics of CP-logic—and in particular, the three-valued interpretation ν_s used in the additional condition imposed by Definition 5.9 for handling negation—is indeed well-defined. Since we defined ν_s as the unique limit of all terminal hypothetical derivation sequences of s, this requires us to show that all such sequence indeed end up in the same limit.

As we recall from Section 2.2, there is a strong duality between three-valued interpretations and pairs (I, J) of two-valued interpretations that are consistent, i.e., for which $I \leq J$. In that chapter, we also defined a corresponding isomorphism τ, that maps each ν to the pair of interpretations (I, J), where I contains all ground atoms p for which $\nu(p) = \mathbf{t}$ and J contains all ground atoms p for which either $\nu(p) = \mathbf{t}$ or $\nu(p) = \mathbf{u}$.

Let us consider a CP-theory C and state s in a C-process. We will denote by $\mathcal{R}(s)$ the set of all CP-events $r \in C$ that have not yet happened in s, i.e., for which there is no ancestor s' of s with $\mathcal{E}(s') = r'$. Let $(\nu_i)_{0 \leq i \leq n}$ be a sequence of three-valued interpretations. It follows directly from the correspondence to pairs of interpretations that $(\nu_i)_i$ is a hypothetical derivation sequence in the state s iff the corresponding sequence of pairs $(I_i, J_i) = \tau(\nu_i)$ satisfies the following conditions:

- For all i, $I_i = \mathcal{I}(s)$;
- $J_0 = I_0$;
- for all $i > 0$, $J_i = J_{i-1} \cup head(r)$, with $r \in \mathcal{R}(s)$ such that $(J, I) \models body(r)$.

In order to characterize the limit reached by such a sequence, we will define a new operator V_s on interpretations, that maps each interpretation J to $J \cup H$, where H is the union of all $heads(r)$ for which $r \in \mathcal{R}(s)$ and, with $I = \mathcal{I}(s)$, $(J, I) \models body(r)$. Clearly, this operator is monotone and has each interpretation J as a postfixpoint. As we recall from Section 2.1.1, this implies that, for each interpretation J, V_s has a unique least fixpoint greater than J. This property can now be used to characterize the limit of a hypothetical derivation sequence.

Proposition 6.1. *Let $(\nu_i)_{0 \leq i \leq n}$ be a terminal hypothetical derivation sequence for a CP-theory C and state s in a C-process. Then $\tau(\nu_n) = (I, J)$, where J is the least fixpoint greater than I of the operator V_s.*

Proof. For all i, let (I, J_i) be $\tau(\nu_i)$. Let J be the least fixpoint greater than I of V_s. Because the sequence $(\nu_i)_{0 \leq i \leq n}$ is terminal, for all CP-events $r \in \mathcal{R}(s)$, if $(J_n, I) \models body(r)$ then $head(r) \subseteq J_n$. It follows that J_n is a fixpoint of V_s. Moreover, J_n is by construction also greater than I. Therefore, $J \leq J_n$ and it suffices to show that also $J_n \leq J$. We prove by induction that the inequality $J_i \leq J$ in fact holds for all $i \in 0..n$. For $J_0 = I$, this is trivial. Assume that, for some $i > 0$, $J_{i-1} \leq J$. There exists a CP-event $r \in \mathcal{R}(s)$ such that $J_i = J_{i-1} \cup head(r)$ and $(J_{i-1}, I) \models body(r)$. Because, by the induction hypothesis, $J_{i-1} \leq J$, this implies that also $(J, I) \models body(r)$. Therefore, $head(r) \in V_s(J) = J$ and we conclude that $J_i \leq J$. □

This result characterizes the limit of any terminal hypothetical derivation sequence in a state s in a way that depends only on s. As such, it shows that all such sequences converge to a unique limit ν_s, namely, the least fixpoint greater than $\mathcal{I}(s)$ of V_s. We have now proven Theorem 5.3 and, therefore, our semantics is indeed well defined.

6.2 CP-logic and LPADs are equivalent

We now establish a link to logic programming, by relating this limit ν_s to the stable operator of a logic program C^s, defined as follows. For a CP-event $r \in C$ that has already happened in s, i.e., $r \notin \mathcal{R}(s)$, we denote by r^s the singleton set of rules $\{h \leftarrow body(r)\}$, where h is the atom that was the result of this event. For every other CP-event r, r^s is the set of all rules $h \leftarrow body(r)$, for which $h \in head_{At}(r)$. The program C^s is now defined as $\cup_{r \in C} r^s$. Recall that in Section 2.2, we defined the semantics of a rule set P by means of a function U_P that maps each pair of interpretations (I, J) to the set of all atoms $head(r)$ for which $(I, J) \models body(r)$.

In order to relate such a program C^s to the operator V_s, we consider the subset $P \subseteq C^s$ that contains all r^s for which $r \in \mathcal{R}(s)$. Let I be $\mathcal{I}(s)$. It is now easy to see that for all J, $V_s(J) = J \cup U_P(J, I)$. Now, obviously $U_{C^s}(J, I) = U_P(J, I) \cup U_{C^s \setminus P}(J, I)$. Because the program $C^s \setminus P$ consists precisely of all r^s for which r has already happened in s, we have that, for all $r \in C^s \setminus P$, $head(r)$ belongs to I. Therefore, for any J, $U_{C^s \setminus P}(J, I) \leq I$. It follows that for all $J \geq I$:

$$V_s(J) = U_P(J, I) \cup J = U_P(J, I) \cup J \cup U_{C^s \setminus P}(J, I) = U_{C^s}(J, I) \cup J.$$

We now use these observations to relate ν_s to the upper stable operator $C^{\uparrow}_{C^s}$; recall that this maps each J to *lfp*$(U_{C^s}(\cdot, J))$ and is also equal to the lower stable operator $C^{\downarrow}_{C^s}$. First, we do this only under the assumption that the pair $(I, C^{\uparrow}_{C^s}(I))$ is consistent.

Proposition 6.2. *Let $(\nu_i)_{0\leq i\leq n}$ be a hypothetical derivation sequence for a CP-theory C and state s in an execution model of C. Let I be $\mathcal{I}(s)$. If $I \leq C^{\uparrow}_{C^s}(I)$, then $\tau(\nu_n) = (I, C^{\uparrow}_{C^s}(I))$.*

Proof. By Proposition 6.1, $\tau(\nu_n) = (I, J)$, with J the least fixpoint greater than I of V_s. Because for each $K \geq I$, $V_s(K) = K \cup U_{C^s}(K, I)$, any fixpoint of $U_{C^s}(\cdot, I)$ greater than I is also a fixpoint of V_s. In particular, by the assumption that $I \leq C^{\uparrow}_{C^s}(I)$, this holds for the least fixpoint $C^{\uparrow}_{C^s}(I)$ of $U_{C^s}(\cdot, I)$. Therefore, $J \leq C^{\uparrow}_{C^s}(I)$. As such, it suffices to show that also $J \geq C^{\uparrow}_{C^s}(I)$. For any $K \geq I$, clearly $V_s(K) \supseteq U_{C^s}(K, I)$. It follows that the least fixpoint greater than I of V_s must be greater than or equal to the least fixpoint greater than I of $U_{C^s}(\cdot, I)$, i.e., indeed $J \geq C^{\uparrow}_{C^s}(I)$. □

We now show that for all $I = \mathcal{I}(s)$ it is indeed the case that $(I, C^{\uparrow}_{C^s}(I))$ is consistent, by proving the following, stronger result. Our formulation of this theorem uses the concept of *prudence*: for a program P, a pair (I, J) is *P-prudent* if I is less than each prefixpoint of $U_P(\cdot, J)$, i.e., if for each K, $U_P(K, J) \leq K$ implies $I \leq K$.

Proposition 6.3. *Let $\mathcal{T}$ be an execution model of C. For each node s, if $I = \mathcal{I}(s)$ and $J = C^{\uparrow}_{C^s}(I)$, then the pair (I, J) is both consistent and prudent.*

Proof. Let $s_0, \ldots, s_n$ be a branch of $\mathcal{T}$, with s_0 the root of $\mathcal{T}$. For each i, let I_i be $\mathcal{I}(s_i)$ and J_i be $C^{\uparrow}_{C^{s_i}}(I_i)$. We will prove by induction that for each i, the pair (I_i, J_i) is both consistent and C^{s_i}-prudent. Because for the root s_0 of $\mathcal{T}$, $I_0 = \{\}$, the pair (I_0, J_0) trivially satisfies this property. Let $i \geq 0$ and assume that the property holds for all $j \leq i$. We construct (I_{i+1}, J_{i+1}) from (I_i, J_i) in two steps, showing that each step preserves consistency and prudence.

First, we go from the pair (I_i, J_i) to (I_{i+1}, J_i). By construction, there exists an $r \in C^{s_{i+1}}$, such that $I_{i+1} = I_i \cup \{head(r)\}$ and $\nu_{s_i}(body(r)) = \mathbf{t}$. By the induction hypothesis and Proposition 6.2, $\tau(\nu_{s_i}) = (I_i, J_i)$, so $(I_i, J_i) \models body(r)$ and $(J_i, I_i) \models body(r)$. Let us now first show that this step preserves consistency, i.e., that $I_{i+1} \leq J_i$. Because already $I_i \leq J_i$, it suffices to show that $head(r) \in J_i$. Because $r \in C^{s_i}$ and $(J_i, I_i) \models body(r)$, we have that $head(r) \in U_{C^{s_i}}(J_i, I_i)$. Because J_i is by construction a fixpoint of $U_{C^{s_i}}(\cdot, I_i)$, this implies that $head(r) \in J_i$. We now show that this step also preserves prudence. Let I be such that $I \geq U_{C^{s_{i+1}}}(I, J_i)$. We need to show that $I \geq I_{i+1}$. Again, since $I_{i+1} = I_i \cup head(r)$ and $I \geq I_i$, it suffices to show that $head(r) \in I$. Because $(I_i, J_i) \models body(r)$ and $I \geq I_i$, we have that $(I, J_i) \models body(r)$ and, therefore, it follows from $I \geq U_{C^{s_{i+1}}}(I, J_i)$ that $head(r) \in I$.

Second, we go from the pair (I_{i+1}, J_i) to (I_{i+1}, J_{i+1}). We first show that this step, too, preserves consistency. Because (I_{i+1}, J_i) is consistent, $(J_{i+1}, I_{i+1}) \geq_p (J_{i+1}, J_i)$ and:

$$J_{i+1} = U_{C^{s_{i+1}}}(J_{i+1}, I_{i+1}) \geq U_{C^{s_{i+1}}}(J_{i+1}, J_i).$$

Because (I_{i+1}, J_i) has already been shown to be $C^{s_{i+1}}$-prudent, this implies that indeed $I_{i+1} \leq J_{i+1}$. We now show that this step also preserves prudence. Let I

be such that $I \geq U_{C^{s_{i+1}}}(I, J_{i+1})$. We need to show that $I \geq I_{i+1}$. It suffices to show that $I \geq U_{C^{s_{i+1}}}(I, J_i)$, because then the result will follow directly from the $C^{s_{i+1}}$-prudence of (I_{i+1}, J_i). Because $I_{i+1} \geq I_i$ and each $C_P^{\uparrow}$-operator is anti-monotone, $J_{i+1} = C^{\uparrow}_{C^{s_{i+1}}}(I_{i+1}) \leq C^{\uparrow}_{C^{s_{i+1}}}(I_i)$. Moreover, because $C^{s_{i+1}} \subseteq C^{s_i}$, $C^{\uparrow}_{C^{s_{i+1}}}(I_i) \leq C^{\uparrow}_{C^{s_i}}(I_i) = J_i$, so $J_{i+1} \leq J_i$. Because $U_{C^{s_{i+1}}}$ is anti-monotone in its second argument, we now have that $U_{C^{s_{i+1}}}(I, J_i) \leq U_{C^{s_{i+1}}}(I, J_{i+1})$ and, since we chose $I \geq U_{C^{s_{i+1}}}(I, J)$, we now indeed find that $I \geq U_{C^{s_{i+1}}}(I, J_i)$. □

Together, the above two propositions imply that, for each node s of an execution model, $\tau(\nu_s)$ is the pair (I, J) for which $I = \mathcal{I}(s)$ and $J = C^{\uparrow}_{C^s}(I)$. Let us now consider a branch $s_0, \ldots, s_n$ of an execution model. Because for all $j \geq i$, $I_j \geq I_i$ and every $C^{\uparrow}_{C^s}$-operator is anti-monotone, the sequence $\tau(\nu_{s_i})_{0 \leq i \leq n} = (I_i, J_i)_{0 \leq i \leq n}$ is increasing with respect to the precision order. We now show that this increasingly precise sequence converges to the well-founded model of C^{s_n}, by showing that, on the one hand, every pair (I_i, J_i) approximates the well-founded model, while, on the other hand, the limit of this sequence is exact, i.e., $I_n = J_n$.

Proposition 6.4. *Let s be a node in an execution model $\mathcal{T}$ of C. Let $I = \mathcal{I}(s)$ and $J = C^{\uparrow}_{C^s}(I)$. For any leaf l that descends from s, if (V, W) is the well-founded model of C^l, then $(I, J) \leq_p (V, W)$.*

Proof. Let $s_0, \ldots, s_n$ be the branch of $\mathcal{T}$ that leads from its root s_0 to the leaf $l = s_n$ and, for each i, let I_i be $\mathcal{I}(s_i)$ and $J_i = C^{\uparrow}_{C^{s_i}}(I_i)$. We now prove by induction that, for each i, $(I_i, J_i) \leq_p (V, W)$. It follows directly from the anti-monotonicity of $C^{\uparrow}_{C^l}$ that, for any interpretation I, if $J = C^{\uparrow}_{C^l}(I)$ and $I \leq V$, then $(I, J) \leq_p (V, W)$. Because $I_0 = \{\} \leq V$, this immediately implies the base case of our induction. Assume now that $(I_i, J_i) \leq_p (V, W)$ and let $r \in C^l$ be such that $I_{i+1} = I_i \cup head(r)$. It suffices to show that $head(r) \in V$. Now, because $(I_i, J_i) \models body(r)$, the induction hypothesis implies that $(V, W) \models body(r)$. Because by construction $V = U_{C^l}(V, W)$, indeed $head(r) \in V$. □

Proposition 6.5. *Let l be a leaf of an execution model of C. Then the pair $(\mathcal{I}(l), \mathcal{I}(l))$ is the well-founded model of C^l.*

Proof. Let (V, W) be the well-founded model of C^l. It is well-known that this implies that $V \leq W$. Now, if we let I be $\mathcal{I}(l)$ and J be $C^{\uparrow}_{C^l}(I)$, then Proposition 6.4 states that $I \leq V \leq W \leq J$. We now show that $I = J$. Because l is a leaf, the principle of sufficient causation implies that, for all $r \in C^l$ such that $I \models body(r)$, $head(r) \in I$. Therefore, $I = U_{C^l}(I, I)$. Because J is the least fixpoint of $U_{C^l}(\cdot, I)$, this implies that $J \leq I$. By Proposition 6.3, also $I \leq J$, so $I = J$. It follows that $I = V = W = J$. □

Having established this relation to the well-founded semantics, we now proceed to show that all execution models of a CP-theory indeed define the same probability distribution and, moreover, that this coincides with the instance based semantics μ_C.

The core concept used to define the instance based semantics is that of a selection. Given an execution model $\mathcal{T}$ of a CP-theory C, we can associate to each leaf l of $\mathcal{T}$ the

set S_l of all those C-selections σ that extend the choices made in the branch leading to l. More formally, if $s_0, \ldots, s_n$ is the branch leading to l (with s_0 the root of $\mathcal{T}$ and $s_n = l$), then $\sigma \in S_l$ iff for all $i < n$, $\mathcal{I}(s_{i+1}) = \mathcal{I}(s_i) \cup \{\sigma^h(\mathcal{E}(s_i))\}$. It is easy to see that, with $L_\mathcal{T}$ the set of all leaves of $\mathcal{T}$, $(S_l)_{l \in L_\mathcal{T}}$ is a partition of the set $\mathcal{S}_C$ of all selections. We now study some properties of these sets of selections S_l. As a technical tool, we first introduce the following notion of an equivalence class of a selection.

Definition 6.1. Let D be a subset of a CP-theory C. Two selections $\sigma, \sigma' \in \mathcal{S}_C$ are *D-equivalent*, denoted $\sigma \equiv_D \sigma'$, iff $\forall r \in D$, $\sigma(r) = \sigma'(r)$. The equivalence class of σ under $\equiv_D$ is denoted as $[\sigma]_D$.

Clearly, any set S_l is equal to the equivalence class $[\sigma]_H$, where $\sigma \in S_l$ and $H = C \setminus \mathcal{R}(l)$ is the set of all rules that have happened leading up to l, i.e., for all $r \in C$, r belongs to H iff l has an ancestor s for which $\mathcal{E}(s) = r$.

Proposition 6.6. *For a CP-theory C, let σ be a C-selection and $D \subseteq C$. Then $P([\sigma]_D) = \prod_{r \in D} \sigma^\alpha(r)$.*

Proof. We begin by making the following calculation:

$$
\begin{aligned}
P([\sigma]_D) &= \sum_{\rho \in [\sigma]_D} P(\rho) = \sum_{\rho \in [\sigma]_D} \prod_{r \in C} \rho^\alpha(r) && \text{(By definition)} \\
&= \sum_{\rho \in [\sigma]_D} \prod_{r \in D} \rho^\alpha(r) \prod_{r \notin D} \rho^\alpha(r) \\
&= \sum_{\rho \in [\sigma]_D} \prod_{r \in D} \sigma^\alpha(r) \prod_{r \notin D} \rho^\alpha(r) && (\forall \rho \in [\sigma]_D : \text{if } r \in D, \text{ then } \rho(r) = \sigma(r)) \\
&= \big(\prod_{r \in D} \sigma^\alpha(r)\big)\big(\sum_{\rho \in [\sigma]_D} \prod_{r \notin D} \rho^\alpha(r)\big) && \text{(Distributivity)}
\end{aligned}
$$

We now show that this last sum $\sum_{\rho \in [\sigma]_D} \prod_{r \notin D} \rho^\alpha(r)$ is in fact equal to 1, which will obviously prove the desired result. For any particular selection σ and $D \subseteq C$, the equivalence class $[\sigma]_D$ is isomorphic to the set $\mathcal{S}_{C \setminus D}$ of all selections for the subtheory $C \setminus D$ of C. Indeed, an isomorphism between these two sets is the function that maps every $(C \setminus D)$-selection τ to the unique C-selection ρ for which $\rho|_D = \sigma|_D$ and $\rho|_{C \setminus D} = \tau$. We therefore find that $\sum_{\rho \in [\sigma]_D} \prod_{r \notin D} \rho^\alpha(r) = \sum_{\tau \in \mathcal{S}_{C \setminus D}} \prod_{r \in C \setminus D} \tau^\alpha(r) = \sum_{\tau \in \mathcal{S}_{C \setminus D}} P(\tau)$. Because for every event r in a CP-theory, $\sum_{(h,\alpha) \in head^*(r)} \alpha = 1$, it is easy to see that, for any CP-theory C', the sum of the probabilities $P(\sigma')$ over *all* C'-selections σ' is always 1. In particular, this must be the case for $C' = C \setminus D$, which now proves the result. □

As previously pointed out, $S_l = [\sigma]_H$ with $\sigma \in S_l$ and $H = C \setminus \mathcal{R}(l)$. The above proposition now shows that $P(S_l) = \prod_{r \in H} \sigma^\alpha(r)$, which is of course equal to the probability $\mathcal{P}(l)$ of the leaf l itself. Having thus established that $\mathcal{P}(l) = P(S_l)$, we now just need to show that, for each $\sigma \in S_l$, the well-founded model of C^σ is $(\mathcal{I}(l), \mathcal{I}(l))$. Proposition 6.7 already shows that $(\mathcal{I}(l), \mathcal{I}(l))$ coincides with the well-founded model of C^l. This is almost the result that we want, but not completely, since

C^σ is not necessarily equal to C^l. In particular, it might be the case that $C^\sigma \subset C^l$, because C^σ contains precisely one instantiation of each CP-event of C, whereas C^l contains *all* possible instantiations of the CP-events that have not yet happened in l. The missing piece of the puzzle is now provided by the following straightforward logic programming result, that we state without proof.

Proposition 6.7. *Let P and P' be logic programs, such that $P \subseteq P'$ and P' has an exact well-founded model (I, I). If P contains all rules $r \in P'$ for which $I \models body(r)$, then (I, I) is also the well-founded model of P.*

It is easy to see that, for any leaf l of an execution model of C and selection $\sigma \in S_l$, C^σ and C^l satisfy the condition of this proposition. This now allows us to prove the desired equivalence, which was previously stated as Theorem 5.8.

Theorem 6.1. *Let $\mathcal{T}$ be an execution model of a CP-theory C. For each interpretation J,*

$$\mu_C(J) = \pi_{\mathcal{T}}(J).$$

Proof. Let $\mathcal{T}$, C and J be as above. Let $L_{\mathcal{T}}(J)$ be the set of all leaves l of $\mathcal{T}$ for which $\mathcal{I}(l) = J$. Then by definition, $\pi_{\mathcal{T}}(J) = \sum_{l \in L_{\mathcal{T}}(J)} \pi_{\mathcal{T}}(l)$. The probability $\mu_C(J)$, on the other hand, is defined as $\sum_{\sigma \in S(J)} P(\sigma)$, where $S(J)$ is the set of all selections σ for which $WFM(C^\sigma) = (J, J)$. Now, every selection σ belongs to precisely one S_l, with l a leaf of $\mathcal{T}$. Moreover, it follows from Proposition 6.7 that for all σ, $WFM(C^\sigma) = (J, J)$ iff there is an $l \in L_{\mathcal{T}}(J)$ such that $\sigma \in S_l$. Therefore, we can partition $S(J)$ into $(S_l)_{l \in L_{\mathcal{T}}(J)}$. By Proposition 6.6, we now have that:

$$\mu_C(J) = \sum_{\sigma \in S(J)} P(\sigma) = \sum_{l \in L_{\mathcal{T}}(J)} \sum_{\sigma \in S_l} P(\sigma) = \sum_{l \in L_{\mathcal{T}}(J)} \mathcal{P}(l) = \pi_{\mathcal{T}}(J).$$

□

For any execution model $\mathcal{T}$ of C, this theorem now characterizes the probability distribution $\pi_{\mathcal{T}}$ in a way that depends only on C and not on $\mathcal{T}$ itself. It follows that, indeed, for all execution models $\mathcal{T}$ and $\mathcal{T}'$ of C, $\pi_{\mathcal{T}} = \pi_{\mathcal{T}'}$, which means that we have now also proven Theorem 5.5 (and, therefore, Theorem 5.1 as well).

6.3 Stratified CP-theories are treated correctly

In this section, we will prove Theorem 5.6, which states that every stratified CP-theory has an execution model which follows its stratification. Recall that a CP-theory is stratified if there exists a mapping λ from its Herbrand base to $\mathbb{N}$, such that for all $h \in head_{At}(r)$ and $b \in body^+_{At}(r)$, $\lambda(h) \geq \lambda(b)$ and for all $h \in head_{At}(r)$ and $b \in body^-_{At}(r)$, $\lambda(h) > \lambda(b)$. In order to prove this result, we will again introduce a mapping κ from C to $\mathbb{N}$. We chose κ to be such that it respects λ and for all $b \in body^-_{At}(r)$, $\kappa(r) > \lambda(b)$. It can easily be seen that for any stratified theory C, it is always possible to find such a κ. Moreover, it is also always possible to construct a κ-process $\mathcal{T}$ that satisfies all the original conditions of Definition 5.4 (execution model—positive case). Indeed, this is simply a matter of executing in each node s a CP-event r

with minimal $\kappa(r)$ among all CP-events whose body is satisfied in s. Indeed, for every child s' of s, it will then be the case that, for all rules r' with $\mathcal{I}(s') \models body(r')$, either $body(r')$ was already satisfied in s or else the atom that was caused by r must belong to $body_{At}(r')$. In both cases, $\kappa(r') \geq \kappa(r)$.

Therefore, it now suffices to show that any such process $\mathcal{T}$ also satisfies the additional condition imposed by Definition 5.9 (execution model—general case). Our proof of this will need some intermediate results.

Let us first recall some properties of hypothetical derivation sequences. Such a sequence makes atoms that were initially $\mathbf{f}$ become $\mathbf{u}$. This implies that such a sequence is decreasing with respect to the knowledge order and increasing with respect to the truth order. Moreover, for all ν_i in such a sequence, the truth value $\nu_i(body(r))$ must therefore be related to whether or not $\mathcal{I}(s) \models body(r)$ in the following way:

- If $\nu_i(body(r)) = \mathbf{t}$, then also $\mathcal{I}(s) \models body(r)$;
- If $\nu_i(body(r)) = \mathbf{f}$, then also $\mathcal{I}(s) \not\models body(r)$;
- If $\nu_i(body(r)) = \mathbf{u}$, then there must be least atom $p \in body_{At}(r)$ that was $\mathbf{f}$ in $\mathcal{I}(s)$ but is $\mathbf{u}$ in ν_i and, moreover:
  - If $\mathcal{I}(s) \models body(r)$, then $p \in body^{-}_{At}(r)$;
  - If $\mathcal{I}(s) \not\models body(r)$, then $p \in body^{+}_{At}(r)$.

Now, let $\mathcal{T}$ be a κ-process that satisfies all the original conditions of Definition 5.4 (execution model—positive case). By definition, for all descendants s' of s, $\kappa(\mathcal{E}(s')) \geq \kappa(\mathcal{E}(s))$. We now show that this implies that for every CP-event r' that could also have happened in s, $\kappa(r') \geq \kappa(\mathcal{E}(s))$.

Proposition 6.8. *Let s be a node of $\mathcal{T}$, let r be $\mathcal{E}(s)$ and let $r' \in \mathcal{R}(s)$ be a CP-event for which $\mathcal{I}(s) \models body(r')$. Then $\kappa(r') \geq \kappa(r)$.*

Proof. Let us assume towards contradiction that there exists a rule $r' \in \mathcal{R}(s)$ such that $\mathcal{I}(s) \models body(r')$ and $\kappa(r') < \kappa(r)$. We first prove by induction that, for each descendant s' of s, it must then still be the case that $\mathcal{I}(s') \models body(r')$. The base case $s' = s$ is trivial. For the induction step, we assume that the property already holds for some descendant s'' of s and consider a child s' of s''. Let r'' be $\mathcal{E}(s'')$. By the induction hypothesis, $\mathcal{I}(s'') \models body(r')$. Therefore, the only way in which it could be possible that $\mathcal{I}(s') \not\models body(r')$ is if r'' causes some atom $h \in head_{At}(r'')$ that also belongs to $body^{-}_{At}(r')$. However, this would imply that $\kappa(r'') < \kappa(r') < \kappa(r)$, which contradicts the fact that $\mathcal{T}$ follows κ. We conclude that for all descendants s' of s, $body(r')$ is indeed still satisfied in $\mathcal{I}(s')$. In particular, this must be the case for every leaf l that can be reached from s. Therefore, r' must happen in some state between s and l. However, because $\mathcal{T}$ follows κ and $\kappa(r') < \kappa(r)$, this cannot be the case and we have our contradiction. □

For a node s, there are only certain CP-events r for which it can be the case that $\nu_s(body(r)) = \mathbf{u}$. Indeed, as we now show, this can only happen for those r whose level $\kappa(r)$ is at least as great as the level of the event $\mathcal{E}(s)$ that actually happens in s.

Proposition 6.9. *Let s be a node of $\mathcal{T}$. For all CP-events $r \in \mathcal{R}(s)$, if $\kappa(r) < \kappa(\mathcal{E}(s))$, then $\nu_s(body(r)) \neq \mathbf{u}$.*

Proof. Let $(\nu_i)_{0 \leq i \leq n}$ be a hypothetical derivation sequence in s. We will show by induction over i that the property in fact holds for all ν_i. Because ν_0 is still two-valued, the base case is trivial. Let us now assume that the property holds for $i-1$. We assume towards contradiction that there is a CP-event $r \in \mathcal{R}(s)$, for which $\kappa(r) < \kappa(\mathcal{E}(s))$ and $\nu_i(body(r)) = \mathbf{u}$. By the induction hypothesis, it is the case that $\nu_{i-1}(body(r)) \neq \mathbf{u}$. This implies that the transition from ν_{i-1} to ν_i must have used some $r' \in \mathcal{R}(s)$ for which $head_{At}(r')$ contains at least one atom that also appears in $body_{At}(r)$. Therefore, $\kappa(r') \leq \kappa(r) < \kappa(\mathcal{E}(s))$. Moreover, it must also be the case that, for this r', $\nu_{i-1}(body(r'))$ is either $\mathbf{t}$ or $\mathbf{u}$. By this induction hypothesis, it cannot be $\mathbf{u}$, so it must by $\mathbf{t}$. However, because a hypothetical derivation sequence is decreasing with respect to the knowledge order, this implies that $\mathcal{I}(s) \models body(r')$. By Proposition 6.8, we then have that $\kappa(r') \geq \kappa(\mathcal{E}(s))$, which contradicts $\kappa(r') < \kappa(\mathcal{E}(s))$. □

We are now ready to finish our proof of Theorem 5.6, by showing that $\mathcal{T}$ must indeed also satisfy the additional condition imposed by Definition 5.9 and is, therefore, an execution model of C.

Proposition 6.10. *A κ-process $\mathcal{T}$ that already satisfies the original conditions of Definition 5.4 must also satisfy the additional condition of Definition 5.9.*

Proof. Let s be a node of a C-process $\mathcal{T}$ that follows κ and satisfies all principles apart from temporal precedence. We need to show that $\nu_s(body(\mathcal{E}(s)) = \mathbf{t}$. Because a hypothetical derivation sequence is decreasing with respect to the knowledge order, the fact that $\mathcal{I}(s) \models body(r)$ implies that $\nu_s(body(\mathcal{E}(s))$ is either $\mathbf{t}$ or $\mathbf{u}$. Let us assume towards contradiction that it is $\mathbf{u}$. There must exists a CP-event r' whose head contains at least one atom also appearing in $body^{-}_{At}(r)$ such that $\nu_s(body(r'))$ is either $\mathbf{t}$ or $\mathbf{u}$. Because then $\kappa(r') < \kappa(r)$, Proposition 6.9 implies that $\nu_s(body(r'))$ cannot be $\mathbf{u}$, so it must be $\mathbf{t}$. However, this can only happen if already $\mathcal{I}(s) \models body(r')$, which contradicts Proposition 6.8. □

This concludes our proof of Theorem 5.6. Since this theorem clearly generalizes Theorem 5.2, we have now proven all theorems stated in Chapter 5

Chapter 7

Conclusions

Our central topic has been the role of constructive processes in knowledge representation. In particular, we have investigated such processes in two distinct settings.

Algebraic study of logics with fixpoint semantics

In the first part of this text, we studied properties of constructive processes in the abstract, algebraic setting of approximation theory. This allowed us to analyze some important knowledge representation concepts in a general, syntax-independent way. From such an algebraic analysis, we could then derive, in an easy and uniform way, concrete results for various fixpoint semantics of a number of different languages. We did this for two topics.

First, we examined *modularity*, by means of the algebraic concept of a stratifiable approximation. We then used our algebraic results to (partly) generalize a number of known splitting results for logic programs (Lifschitz and Turner 1994; Eiter, Gottlob, and Mannila 1997), open logic programs (Verbaeten, Denecker, and Schreye 2000), ID-logic (Denecker and Ternovska 2004), autoepistemic logic (Gelfond and Przymusinska 1992; Niemelä and Rintanen 1994), and default logic (Turner 1996).

Second, we also studied the topic of *predicate introduction*, by means of the algebraic concept of fixpoint extension. In the case of logic programming, our results significantly generalize an earlier result by Van Gelder (Van Gelder 1993), as well as a result by (Dix and Müller 1994) and certain restricted forms of fold/unfold transformations (Aravindan and Dung 1995). In the case of autoepistemic logic, we presented a transformation to reduce the nesting depth of the modal operator, which offers an alternative (avoiding a blowup in the size of the theory at the cost of enlarging the alphabet) to the transformation presented in (Marek and Truszczyński 1991).

The main contribution of the work presented in this first part is that we have shown approximation theory to be a viable way of studying properties of knowledge representation languages with a fixpoint semantics in a general way, without committing to a single specific formalism. Indeed, for both topics we investigated, we have demonstrated that different results, proven independently in the literature, can be de-

rived from a single theorem in approximation theory, with relative ease. The work of (Truszczyński 2006) on algebraically characterizing strong and uniform equivalence also fits into this strand of research. Along the way, we have also generalized existing results for particular logics in several ways, as summarized in the preceding paragraphs. Mostly, these were rather straightforward generalizations, such as extending an existing result to a more general syntax, a larger class of theories, or a different semantics. Our most significant new applied result has been achieved in the context of predicate introduction for logic programs, where our extension to recursively defined new predicates allowed us to come up with a method of eliminating universal quantifiers in rule bodies.

This work could still be extended in several ways. First, it could be examined if and how other interesting languages fit into the framework of approximation theory. In particular, within the field of answer set programming, the original language of normal logic programs under the stable semantics has been significantly extended, as in, e.g., (Lifschitz, Tang, and Turner 1999). Currently, these extensions fall outside the scope of approximation theory and, therefore, none of our results apply to them. Second, there are of course also other interesting knowledge representation properties that could be investigated in the setting of approximation theory. For instance, the topic of program specialization (Leuschel 1997) seems to be interesting in this respect. Another interesting possibility is examining the fold/unfold transformations, that fall outside the scope of our predicate introduction results.

Constructive processes and causality

In the second part of this text, we showed that constructive processes also play an important role when dealing with causality. We started by considering the informal meaning of causal statements such as “pneumonia causes cast pain” and showed that this implicitly refers to a dynamic evolution, that can be described as a constructive process. By considering also non-deterministic causal statements, such as “pneumonia might cause chest pain”, where we quantify the uncertainty with probabilities, we ended up with the probabilistic modelling language of CP-logic. As suggested by our informal reading of such statements, we defined the semantics of theories in this language by means of certain constructive probabilistic processes, called the execution models of a theory. An important result was that all execution models of the same theory generate precisely the same probability distribution over their possible final states.

An interesting property of CP-logic is that it allows causal events to be described in quite a flexible and fine-grained way. As we have shown, this gives the language certain representation advantages over Bayesian networks, e.g., when it comes to modelling effects with a number of independent possible causes or cyclic causal relations. We have also related CP-logic to the field of logic programming, by first defining a probabilistic extension of disjunctive logic programs, called logic programs with annotated disjunctions, and then showing that this is equivalent to CP-logic. This result allows a new and appealing informal interpretation of (probabilistic) logic programming constructs in terms of causal events.

The main contribution of this work has been to develop a language whose syn-

tax and semantics follow naturally from properties that are inherent to a certain kind of causal statement. As such, CP-logic has primarily served as a tool for analyzing the nature of these statements and their role in probabilistic modelling. While we have shown that it already has some interesting properties when compared to Bayesian networks, future work could still make CP-logic more practically applicable; most notably, the language itself could be extended in a number of obvious ways, it should be investigated how CP-theories can be integrated with other forms of probabilistic knowledge, and inference methods for CP-logic should be studied.

Bibliography

Apt, K., H. Blair, and A. Walker (1988). Towards a theory of Declarative Knowledge. In J. Minker (Ed.), *Foundations of Deductive Databases and Logic Programming*, pp. 89–148. Morgan Kaufmann.

Aravindan, C. and P. M. Dung (1995). On the correctness of unfold/fold transformation of normal and extended logic programs. *Journal of Logic Programming 24*(3), 201–217.

Bacchus, F. (1993). Using first-order probability logic for the construction of Bayesian networks. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence, UAI'93*, pp. 219–226.

Balduccini, M. and M. Gelfond (2003). Diagnostic reasoning with A-Prolog. *Theory and Practice of Logic Programming (TPLP) 3*(4-5), 425–461.

Baral, C., M. Gelfond, and N. Rushton (2004). Probabilistic reasoning with answer sets. In *Proc. of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-7)*, Volume 2923 of *Lecture notes in artifical intelligence (LNAI)*, pp. 21–33. Springer-Verlag.

Baral, C. and V. Subrahmanian (1991). Duality between alternative semantics of logic programs and nonmonotonic formalisms. In A. Nerode, W. Marek, and V. Subrahmanian (Eds.), *Intl. Workshop on Logic Programming and Nonmonotonic Reasoning*, Washington DC., pp. 69–86. MIT Press.

Belnap, N. D. (1977). A useful four-valued logic. In *Modern uses of multiple-valued logic*, pp. 5–37. Reidel, Dordrecht, NL.

Blockeel, H. and W. Meert (2007). Two novel methods for learning logic programs with annotated disjunctions. In *Proceedings of the 16th International Conference on Inductive Logic Programming, ILP'06*, Lecture Notes in Computer Science. to appear.

Bonatti, P. (1995). Autoepistemic logics as a unifying framework for the semantics of logic programs. *Journal of Logic Programming 22*, 91–149.

Breese, J. (1992). Construction of belief and decision networks. *Computational intelligence 8*(4), 624–647.

Clark, K. L. (1978). Negation as failure. In H. Gallaire and J. Minker (Eds.), *Logic and Databases*, pp. 293–322. Plenum Press.

Costa, V. S., D. Page, M. Qazi, and J. Cussens (2003). CLP(BN): Constraint logic programming for probabilistic knowledge. In *Proceedings of the Nineteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-2003)*, pp. 517–524. Morgan Kaufmann.

Cussens, J. (2000). Stochastic logic programs: Sampling, inference and applications. In *Proceedings of the Sixteenth Annual Conference on Uncertainty in Artificial Intelligence*, pp. 115–122. Morgan Kaufmann.

De Raedt, L., A. Kimmig, and H. Toivonen (2007). ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI'07*, pp. 2462–2467.

Dekhtyar, A. and V. Subrahmanian (2000). Hybrid probabilistic programs. *Journal of Logic Programming 43*(3), 187–250.

Dell'Armi, T., W. Faber, G. Ielpa, C. Koch, N. Leone, S. Perri, and G. Pfeifer (2001). System description: DLV. In T. Eiter, W. Faber, and M. Truszczyński (Eds.), *LPNMR*, Volume 2173 of *Lecture Notes in Computer Science*, pp. 424–428. Springer.

Denecker, M., V. Marek, and M. Truszczynski (1998). Fixpoint 3-valued semantics for autoepistemic logic. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 840–845. MIT Press / AAAI-Press.

Denecker, M., V. Marek, and M. Truszczyński (2000). Approximating operators, stable operators, well-founded fixpoints and applications in non-monotonic reasoning. In *Logic-based Artificial Intelligence*, The Kluwer International Series in Engineering and Computer Science, pp. 127–144. Kluwer Academic Publishers, Boston.

Denecker, M., V. Marek, and M. Truszczyński (2003, January). Uniform semantic treatment of default and autoepistemic logics. *Artificial Intelligence 143*(1), 79–122.

Denecker, M., V. Marek, and M. Truszczyński (2004). Ultimate approximation and its application in nonmonotonic knowledge representation systems. *Information and Computation 192*(1), 84–121.

Denecker, M. and E. Ternovska (2004). A logic of non-monotone inductive definitions and its modularity properties. In V. Lifschitz and I. Niemelä (Eds.), *Seventh International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'04*.

Dix, J. (1995). A classification theory of semantics of normal logic programs: II. weak properties. *Fundamenta Informaticae 22*(3), 257–288.

Dix, J. and M. Müller (1994). Partial evaluation and relevance for approximations of stable semantics. In Z. W. Ras and M. Zemankova (Eds.), *ISMIS*, Volume 869 of *Lecture Notes in Computer Science*, pp. 511–520. Springer.

Eiter, T., G. Gottlob, and H. Mannila (1997). Disjunctive datalog. *ACM Transactions on Database Systems (TODS) 22*, 364–418.

Erdoğan, S. and V. Lifschitz (2004). Definitions in Answer Set Programming. In *Proc. Logic Programming and Non Monotonic Reasoning, LPNMR'04*, Volume 2923 of *LNAI*, pp. 185–197. Springer-Verlag.

Etherington, D. (1988). *Reasoning with incomplete information*. Research notes in Artificial Intelligence. Morgan Kaufmann.

Fierens, D., H. Blockeel, M. Bruynooghe, and J. Ramon (2005). Logical Bayesian networks and their relation to other probabilistic logical models. In *Proceedings of the 15th International Conference on Inductive Logic Programming, ILP'05*, Volume 3625 of *Lecture Notes in Computer Science*, pp. 121–135. Springer.

Finzi, A. and T. Lukasiewicz (2003). Structure-based causes and explanations in the independent choice logic. In *Proceedings of the 19h Conference on Uncertainty in Artificial Intelligence, UAI'03*.

Fitting, M. (1985). A Kripke-Kleene Semantics for Logic Programs. *Journal of Logic Programming 2*(4), 295–312.

Fitting, M. (1989). Negation as refutation. In *Fourth Annual Symposium on Logic in Computer Science (LICS'89)*, pp. 63–70. IEEE Press.

Fitting, M. (2002). Fixpoint semantics for logic programming - a survey. *Theoretical Computer Science 278*, 25–51.

Gallagher, J. P. (1993, June). Specialisation of logic programs: A tutorial. In *ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM '93)*, pp. pages 88–98.

Gardner, P. and J. Shepherdson (1991). Unfold/fold transformations of logic programs. In J.-L. Lassez and G. Plotkin (Eds.), *Computation logic: Essays in honor of Alan Robinson*. MIT Press.

Gelfond, M. (1987). On Stratified Autoepistemic Theories. In *National Conference on Artificial Intelligence (AAAI'87)*, pp. 207–211. Morgan Kaufman.

Gelfond, M. and V. Lifschitz (1988). The stable model semantics for logic programming. In *International Joint Conference and Symposium on Logic Programming (JICSLP'88)*, pp. 1070–1080. MIT Press.

Gelfond, M. and H. Przymusinska (1992). On consistency and completeness of autoepistemic theories. *Fundamenta Informaticae 16*(1), 59–92.

Gelfond, M. and H. Przymusinska (1996). Towards a theory of elaboration tolerance: Logic programming approach. *Journal on Software and Knowledge Engineering 6*(1), 89–112.

Getoor, L., N. Friedman, D. Koller, and A. Pfeffer (2001). Learning probabilistic relational models. In S. Dzeroski and N. Lavrac (Eds.), *Relational Data Mining*, pp. 7–34. Springer-Verlag.

Ginsberg, M. (1988). Multivalued logics: A uniform approach to reasoning in artificial intelligence. *Computational Intelligence 4*, 265–316.

Halpern, J. and J. Pearl (2001a). Causes and explanations: A structural model approach – part I: Causes. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence, UAI'01*.

Halpern, J. and J. Pearl (2001b). Causes and explanations: A structural model approach – part II: Explanations. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence, UAI'01*.

Halpern, J. and M. Tuttle (1993). Knowledge, probability, and adversaries. *Journal of the ACM 40*, 917–960.

Jaeger, M. (1997). Relational Bayesian networks. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI-97)*.

Kersting, K. and L. D. Raedt (2000). Bayesian logic programs. In J. Cussens and A. Frisch (Eds.), *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pp. 138–155.

Kersting, K. and L. D. Raedt (2001). Bayesian logic programs. Technical Report 151, Institute for Computer Science, University of Freiburg, Germany.

Kersting, K. and L. D. Raedt (2007). Bayesian logic programming: Theory and tool. In L. Getoor and B. Taskar (Eds.), *An Introduction to Statistical Relational Learning*. MIT Press. To appear.

Konolige, K. (1987). On the relation between default and autoepistemic logic. In M. L. Ginsberg (Ed.), *Readings in Nonmonotonic Reasoning*, pp. 195–226. Los Altos, CA: Kaufmann.

Konolige, K. (1988). On the relation between default and autoepistemic logic. *Artificial Intelligence 35*, 343–382.

Lakshmanan, L. and F. Sadri (1994). Probabilistic deductive databases. In M. Bruynooghe (Ed.), *Proceedings of the International Symposium on Logic Programming, ILPS'94*, pp. 254–268. MIT Press.

Leone, N., P. Rullo, and F. Scarcello (1995). Declarative and fixpoints characterizations of disjunctive stable models. In *Proceedings of the International Logic Programming Symposium, ILPS'95*, pp. 399–413. MIT Press.

Leuschel, M. (1997). Advanced techniques for logic program specialisation. *AI Communications 10*(2), 127–128.

Lifschitz, V., L. R. Tang, and H. Turner (1999). Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence 25*(3-4), 369–389.

Lifschitz, V. and H. Turner (1994). Splitting a logic program. In P. V. Hentenryck (Ed.), *International Conference on Logic Programming (ICLP'94)*, pp. 23–37. MIT Press.

Lloyd, J. and R. Topor (1984). Making Prolog more expressive. *Journal of Logic Programming 1*(3), 225–240.

Lukasiewicz, T. (2001). Fixpoint characterizations for many-valued disjunctive logic programs. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*, Volume 2173 of *Lecture Notes in Artificial Intelligence*, pp. 336–350. Springer-Verlag.

Maher, M. J. (1993). A transformation system for deductive database modules with perfect model semantics. *Theoretical Computer Science 110*, 377–403.

Marek, V. and M. Truszczyński (1989). Stable semantics for logic programs and default reasoning. In E. Lust and R. Overbeek (Eds.), *Proceedings of the North American Conference on Logic Programming and Non-monotonic Reasoning*, pp. 243–257.

Marek, V. and M. Truszczyński (1991). Autoepistemic logic. *Journal of the ACM 38*(3), 588–619.

Mariën, M., R. Mitra, M. Denecker, and M. Bruynooghe (2005). Satisfiability checking for PC(ID). In G. Sutcliffe and A. Voronkov (Eds.), *Proceedings of the 12th International Conference on Logic for Programming, Artificial International, and Reasoning, LPAR'05*, Volume 3835 of *Lecture Notes in Computer Science*, pp. 565–579. Springer.

McCain, N. (1997). *Causality in Commonsense Reasoning about Actions*. Ph. D. thesis, University of Texas at Austin.

McCain, N. and H. Turner (1996). Causal theories of action and change. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference (13th AAAI/8th IAAI)*, pp. 460–465. AAAI Press.

Meyer, J.-J. and W. van der Hoek (1995). *Epistemic Logic for Computer Science and Artificial Intelligence*. Cambridge University Press.

Moore, R. (1984). Possible-world semantics for autoepistemic logic. In *Proc. of the Non-Monotonic Reasoning Workshop*, Mohonk, N.Y, pp. 344–354. AAAI Press.

Muggleton, S. (2000). Learning stochastic logic programs. *Electronic Transactions in Artificial Intelligence 5*(041), 141–153.

Ng, R. and V. Subrahmanian (1992). Probabilistic logic programming. *Information and Computation 101*(2), 150–201.

Ngo, L. and P. Haddawy (1997). Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science 171*(1–2), 147–177.

Niemelä, I. and J. Rintanen (1994). On the impact of stratification on the complexity of nonmonotonic reasoning. *Journal of Applied Non-Classical Logics 4*(2).

Niemelä, I., P. Simons, and T. Syrjänen (2000, April). Smodels: a system for answer set programming. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning, NMR'00*, Breckenridge, Colorado, USA.

Pearl, J. (2000). *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

Pelov, N. and M. Truszczyński (2004). Semantics of disjunctive programs with monotone aggregates - an operator-based approach. In J. P. Delgrande and T. Schaub (Eds.), *Proceedings of the 10th International Workshop on Non-Monotonic Reasoning, NMR'04, Whistler, Canada, June 6-8, 2004, Proceedings*, pp. 327–334.

Pettorossi, A. and M. Proietti (1994). Transformations of logic programs: Foundations and techniques. *Journal of Logic Programming 19, 20*.

Pinto, J. and R. Reiter (1993). Temporal reasoning in logic programming: A case for the situation calculus. In *Proc. of the International Conference on Logic Programming*, pp. 203–221.

Poole, D. (1993). Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence 64 64*(1), 81–129.

Poole, D. (1997). The Independent Choice Logic for modelling multiple agents under uncertainty. *Artificial Intelligence 94*(1-2), 7–56.

Przymusinski, T. (1998). Every logic program has a natural stratification and an iterated least fixed point model. In *Proceedings of the 8th Symposium on Principles of Database Systems, PODS'98*, pp. 11–21.

Przymusinski, T. C. (1991). Stable semantics for disjunctive programs. *New Generation Computing 3/4*, 401–424.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence 13*(1–2), 81–132.

Riguzzi, F. (2004, September). Learning logic programs with annotated disjunctions. In A. Srinivasan and R. King (Eds.), *14th Internation Conference on Inductive Logic Programming (ILP2004), Porto, 6-8 September 2004*, Heidelberg, Germany, pp. 270–287. Springer Verlag.

Sakama, C. and K. Inoue (1994). An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of automated reasoning 13*(1), 145–172.

Sato, T. and Y. Kameya (1997). PRISM: A language for symbolic-statistical modeling. In *Proceedings of the International Joint Conferences on Artificial Intelligence, IJCAI'97*, pp. 1330–1335.

Schrijvers, T. and A. Serebrenik (2004). Improving Prolog programs: Refactoring for Prolog. In *Logic Programming, 20th International Conference, ICLP'04, Proceedings*, Volume 3132 of *Lecture Notes in Computer Science*, pp. 58–72.

Seki, H. (1993). Unfold/fold transformation of logic programs for the well-founded semantics. *Journal of Logic Programming 6*, 5–23.

Sneyers, J., J. Vennekens, and D. De Schreye (2006). Probabilistic-logical modeling of music. In *Practical Aspects of Declarative Languages, 8th International Symposium, PADL'06, Proceedings*, Volume 3819 of *LNCS*, pp. 60–72. Springer Verlag.

Tamaki, H. and T. Sato (1984). Unfold/fold transformation of logic programs. In *Proceedings of the Second International Conference on Logic Programming, ICLP'84*, pp. 127–138.

Truszczyński, M. (2006). Strong and uniform equivalence of nonmonotonic theories — an algebraic approach. In *Principles of Knowledge Representation and Reasoning, Proceedings of the Tenth International Conference, KR'06*.

Turner, H. (1996). Splitting a default theory. In *Proc. Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pp. 645–651. AAAI Press.

Van Belleghem, K., M. Denecker, and D. De Schreye (1997). On the Relation between Situation Calculus and Event Calculus. *Journal of Logic Programming, special issue on Reasoning about Actions and Change 31*(1-3), 3–37.

Van Gelder, A. (1993). The alternating fixpoint of logic programs with negation. *Journal of Computer and System Sciences 47*(1), 185–221.

Vennekens, J. and M. Denecker (2005). An algebraic account of modularity in ID-logic. In *Logic Programming and Nonmonotonic Reasoning, LPNMR'05, Proceedings*, Volume 3662 of *Lecture Notes in Computer Science*, pp. 291–303. Springer Verlag.

Vennekens, J., M. Denecker, and M. Bruynooghe (2006). Representing causal information about a probabilistic process. In *Logics in Artificial Intelligence, 10th European Conference, JELIA'06, Proceedings*, Volume 4160 of *Lecture Notes in Computer Science*, pp. 452–464. Springer.

Vennekens, J., D. Gilis, and M. Denecker (2004a). Splitting an operator: An algebraic modularity result and its application to auto-epistemic logic. In *Proceedings of the 10th International Workshop on Non-Monotonic Reasoning, NMR'04*, pp. 400–408.

Vennekens, J., D. Gilis, and M. Denecker (2004b). Splitting an operator: An algebraic modularity result and its applications to logic programming. In *Logic Programming, 20th International Conference, ICLP'04, Proceedings*, Volume 3132 of *Lecture Notes in Computer Science*, pp. 195–209. Springer.

Vennekens, J., D. Gilis, and M. Denecker (2006, October). Splitting an operator: Algebraic modularity results for logics with fixpoints semantics. *ACM Transactions on Computational Logic (ACM TOCL) 7*(4), 765–797.

Vennekens, J., M. Mariën, J. Wittocx, and M. Denecker (2007a). Predicate introduction for logics with a fixpoint semantics. part I: Logic programming. To appear in *Fundamentae Informaticae*.

Vennekens, J., M. Mariën, J. Wittocx, and M. Denecker (2007b). Predicate introduction for logics with a fixpoint semantics. part II: Autoepistemic logic. To appear in *Fundamentae Informaticae*.

Vennekens, J. and S. Verbaeten (2003). A general view on probabilistic logic programming. In *Proceedings 15th Belgian-Dutch Conference on Artificial Intelligence*, pp. 299–306.

Vennekens, J., S. Verbaeten, and M. Bruynooghe (2004). Logic programs with annotated disjunctions. In *Logic Programming, 20th International Conference, ICLP'04, Proceedings*, Volume 3132 of *Lecture Notes in Computer Science*, pp. 431–445. Springer.

Verbaeten, S., M. Denecker, and D. D. Schreye (2000). Compositionality of normal open logic programs. *Journal of Logic Programming 41*, 151–183.

Voronkov, A. (1992). Logic programming with bounded quantifiers. In *Proceedings of the First Russian Conference on Logic Programming, RCLP'91*, Volume 592 of *LNAI*, pp. 486–514. Springer-Verlag.

Wittocx, J., J. Vennekens, M. Mariën, M. Denecker, and M. Bruynooghe (2006). Predicate introduction under stable and well-founded semantics. In *Logic Programming, 22nd International Conference, ICLP'06, Seattle, WA, USA, August 17-20, 2006, Proceedings*, Volume 4079 of *Lecture Notes in Computer Science*, pp. 242–256. Springer.

Biography

Joost Vennekens was born on 27 September 1980 in Turnhout, Belgium. In 1998, he graduated from high school at the V.S.L.S. Westerlo and went to study Informatics (licentiaat informatica) at the K.U. Leuven, from which he graduated *magna cum laude* on the 6th of July, 2002. Since then, he has been working as a PhD-student at the research group DTAI of the Department of Computer Science at the K.U. Leuven, first under the supervision of Prof. D. De Schreye and later also under that of Prof. M. Denecker.

Publication List

Articles in international reviewed journals

- J. Vennekens, D. Gilis, and M. Denecker, Splitting an operator: Algebraic modularity results for logics with fixpoints semantics, ACM Transactions on Computational Logic 7 (4), pp. 765-797, October, 2006.

- J. Vennekens, M. Mariën, J. Wittocx, and M. Denecker, Predicate introduction for logics with a fixpoint semantics. Part I: Logic programming, Fundamenta Informaticae, 2007, to appear.

- J. Vennekens, M. Mariën, J. Wittocx, and M. Denecker, Predicate introduction for logics with a fixpoint semantics. Part II: Autoepistemic logic, Fundamenta Informaticae, 2007, to appear.

Contributions at international conferences, published in Springer's LNCS series

- J. Wittocx, J. Vennekens, M. Mariën, M. Denecker, and M. Bruynooghe, Predicate introduction under stable and well-founded semantics, Logic Programming, 22nd International Conference, ICLP 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings (Etalle, S. and Truszczynski, M., eds.), vol 4079, Lecture Notes in Computer Science, pp. 242-256, 2006.

- J. Vennekens, M. Denecker, and M. Bruynooghe, Representing causal information about a probabilistic process, Logics in Artificial Intelligence, 10th European Conference, JELIA 2006, Proceedings (Fisher, M. and van der Hoek, W. and Konev, B. and Lisitsa, A., eds.), vol 4160, Lecture Notes in Computer Science, pp. 452-464, 2006.

- J. Sneyers, J. Vennekens, and D. De Schreye, Probabilistic-logical modeling of music, Practical Aspects of Declarative Languages, 8th International Symposium, PADL 2006, Proceedings (Van Hentenryck, P., ed.), vol 3819, Lecture Notes in Computer Science, pp. 60-72, 2006.

- J. Vennekens, and M. Denecker, An algebraic account of modularity in ID-logic, Logic Programming and Nonmonotonic Reasoning, LPNMR 2005, Proceedings (Baral, C. and Greco, G. and Leone, N. and Terracina, G., eds.), vol 3662, Lecture Notes in Computer Science, pp. 291-303, 2005.

- J. Vennekens, S. Verbaeten, and M. Bruynooghe, Logic programs with annotated disjunctions, Logic Programming, 20th International Conference, ICLP 2004, Proceedings (Demoen, B. and Lifschitz, V., eds.), vol 3132, Lecture Notes in Computer Science, pp. 431-445, 2004.

- J. Vennekens, D. Gilis, and M. Denecker, Splitting an operator: An algebraic modularity result and its applications to logic programming, Logic Programming, 20th International Conference, ICLP 2004, Proceedings (Lifschitz, V. and Demoen, B., eds.), vol 3132, Lecture Notes in Computer Science, pp. 195-209, 2004.

Contributions at international conferences, published elsewhere

- J. Vennekens, and M. Denecker, Analysing the structure of definitions in ID-logic, Proceedings of the 11th International Workshop on Non-monotonic Reasoning (Dix, J. and Hunter, A., eds.), pp. 183-190, 2006.

- J. Vennekens, M. Denecker, and M. Bruynooghe, Extending the role of causality in probabilistic modeling, Proceedings of the 11th International Workshop on Non-monotonic Reasoning (Dix J. and Hunter, A., eds.), pp. 183-190, 2006.

- J. Vennekens, and M. Denecker, An algebraic account of modularity in ID-logic, Answer Set Programming: Advances in Theory and Implementation (De Vos, M. and Provetti, A., eds.), pp. 57-69, 2005.

- J. Vennekens, S. Verbaeten, and M. Bruynooghe, Logic programs with annotated disjunctions, Proceedings of the 10th International Workshop on Non-Monotonic Reasoning (Delgrande, J.P. and Schaub, T., eds.), pp. 409-415, 2004.

- J. Vennekens, D. Gilis, and M. Denecker, Splitting an operator: An algebraic modularity result and its application to auto-epistemic logic, Proceedings of the 10th International Workshop on Non-Monotonic Reasoning (Delgrande, J.P. and Schaub, T., eds.), pp. 400-408, 2004.

- J. Vennekens, and S. Verbaeten, A general view on probabilistic logic programming, Proceedings of the 15th Belgian-Dutch Conference on Artificial Intelligence (Heskes, T. and Lucas, P. and Vuurpijl, L. and Wiegerinck, W., eds.), pp. 299-306, 2003.

Contributions at international conferences, not published or only as abstract

- J. Vennekens, M. Denecker, and M. Bruynooghe, Representing causal information about a probabilistic process, 18th Belgium-Netherlands Conference on Artificial Intelligence, BNAIC 2006, Namur, Belgium, October 5-6, 2006.
- J. Vennekens, and M. Denecker, An analysis of dependencies in ID-logic, Dagstuhl seminar 05171: Nonmonotonic Reasoning, Answer Set Programming and Constraints, Schloss Dagstuhl, Wadern, Germany, April 25-29, 2005.
- J. Vennekens, S. Verbaeten, and M. Bruynooghe, Logic programs with annotated disjunctions, 5th ”Freiburg, Leuven and Friends” Workshop on Machine Learning, FLF-04, Hinterzarten, Germany, March 8-10, 2004.

Technical reports

- M. Denecker, and J. Vennekens, ID-logic in perspective, Department of Computer Science, K.U.Leuven, Report CW 410, Leuven, Belgium, April, 2005.
- J. Vennekens, and S. Verbaeten, Logic programs with annotated disjunctions, Department of Computer Science, K.U.Leuven, Report CW 368, Leuven, Belgium, September, 2003.