INFORMATIK
BERICHTE

354 – 9/2009

Relational Approaches to Knowledge Representation and Learning

Workshop at the 32nd Annual German Conference on Artificial Intelligence, KI-2009
Paderborn, Germany, September 15, 2009
Proceedings

Christoph Beierle, Gabriele Kern-Isberner (Eds.)

Fakultät für Mathematik und Informatik
Postfach 940
D-58084 Hagen

Christoph Beierle, Gabriele Kern-Isberner (Eds.)

Relational Approaches to Knowledge Representation and Learning

Workshop at the 32nd Annual German Conference
on Artificial Intelligence, KI-2009
Paderborn, Germany, September 15, 2009
Proceedings

Workshop Organization

Workshop Organizers and Co-Chairs

Gabriele Kern-Isberner	Technische Universität Dortmund, Germany
Christoph Beierle	FernUniversität in Hagen, Germany

Program Committee

Salem Benferhat	Université d'Artois, Lens, France
Gerd Brewka	Universität Leipzig, Germany
James P. Delgrande	Simon Fraser University, Burnaby, BC, Canada
Jürgen Dix	TU Clausthal-Zellerfeld, Germany
Eduardo Fermé	Universidade da Madeira, Portugal
Andreas Herzig	Université Paul Sabatier, Toulouse, France
Pascal Hitzler	Universität Karlsruhe (TH), Germany
Antonis C. Kakas	University of Cyprus, Cyprus
Kristian Kersting	Fraunhofer IAIS, Universität Bonn, Germany
Thomas Lukasiewicz	University of Oxford, UK
Torsten Schaub	Universität Potsdam, Germany
Emil Weydert	University of Luxembourg, Luxembourg

Preface

Knowledge representation encompasses a variety of methods and formalisms to encode and process all types of knowledge, belief, and information. It provides the theoretical foundation for rational and intelligent behaviour in real environments, focusing on topics like default logics and uncertain reasoning, belief change, ontologies, and argumentation, among many others. Moreover, in a thematical respect, knowledge representation is closely related to the areas of machine learning and knowledge discovery the methods of which allow the acquisition of useful information to build up knowledge bases.

Knowledge representation has made substantial progress over the last decade by devising sophisticated methods for inference and reasoning. Nevertheless, the connection to learning still holds undeveloped potential in methodological and technical respects which might be crucial for practical applications. Furthermore, the handling of relational information, i.e. the explicit representation of knowledge about objects and its linking to knowledge about classes, is still a challenge for many subareas of knowledge representation. Ontologies, logic programming and probabilistic relational models are just some important examples of areas of research that address both of these points.

This volume contains the contributions that were presented at the Workshop *Relational Approaches to Knowledge Representation and Learning* on September 15, 2009, in Paderborn, Germany, co-located with the 32nd Annual German Conference on AI (KI-2009), and organized by the Special Interest Group on Knowledge Representation and Reasoning of the Gesellschaft für Informatik (*GI-Fachgrupppe Wissensrepräsentation und Schließen*). The particular focus of this workshop was to strengthen the connection between knowledge representation and learning by focusing on relational and first-order approaches to all areas of knowledge representation and learning.

The workshop started with an invited talk *Relevance, Conditionals, and Defeasible Reasoning* by James Delgrande. He investigates the notion of relevance in the context of defeasible reasoning, and presents an approach for incorporating irrelevant properties in a conditional knowledge base. It is argued that this approach exactly captures defeasible reasoning with commonsense normative conditionals.

The beliefs of an agent can be represented by a designated predicate in a selfreferential first-order language. However, such first-order theries often lead to pardoxes. In his paper *$Log_A\mathbf{B}$: An Algebraic Logic of Belief*, Haythem Ismail develops a family of algebraic logics of beliefs that is almost as expressive as first-order theories, but at the same time weak enough to avoid pardoxes of self-reference.

Probabilistic logic is the general topic of the following three papers. With the

origins of probabilistic logic based on propositional logic, the introduction of probabilistic graphical models, in particular the popular Markov and Bayesian networks, enabled practical applications and spurred the research efforts in this area.

In his contribution *First-Order Probabilistic Conditional Logic - Introduction and Representation*, Jens Fisseler presents a first-order extension of a propositional probabilistic representation formalism, allowing in particular the representation of prababilistic *if-then* rules. The semantics employs the principle of maximum entropy, selecting a model that is as unbiased as possible. In order to tame the complexity of the resulting optimization problem to be solved, sufficient syntactic criteria for its simplification are developed.

Currently, the most prominent approaches to lifting propositional probabilistic logic to the first-order case, are Bayesian logic programs (BLP) and Markov logic networks (MLN). Both BLPs and MLNs as well as a new approach for using maximum entropy methods in a relational context are supported by the KReator toolbox that aims at providing a common and simple interface for working with different relational probabilistic approaches. This integrated development environment is presented by Marc Finthammer, Sebastian Loh, and Matthias Thimm in their contribution *Towards a Toolbox for Relational Probabilistic Knowledge Representation, Reasoning, and Learning.*

In his paper *Representing Statistical Information and Degrees of Belief in First-Order Probabilistic Conditional Logic*, Matthias Thimm proposes a formal semantics for first-order probabilistic conditionals that matches common sense and avoids ambiguities between statistical and subjective interpretations. Moreover, he shows how the principle of maximum entropy can also be applied in this framework, and proves formal properties of the resulting inference operator.

Finally, the last two papers deal with approaches based on logic programming. The paper *Reinforcement Learning for Golog Programs* by Daniel Beck and Gerhard Lakemeyer presents an approach of using the action language Golog to constrain the action state space to be explored in a reinforcement learning situation. The authors develop a Golog dialect using a semi-Markov Decision Process representation, and give a completely declarative specification of a learning Golog interpreter.

In general, conflicts may arise when pieces of information depend on each other, and so, Patrick Krümpelmann makes dependencies the atomic concept to study conflicts. In *Towards Dependency Semantics for Conflict Handling in Logic Programs*, he presents a formal framework for conflict resolution that is based on dependencies and involves consequences and preferences. As an application, he shows how this can be related to answer set semantics and the causal rejection principle.

We would like to thank all Program Committee members as well as the additional external reviewer Maurício Reis for detailed and high-quality reviews for all submitted papers. Many thanks also to the organizers of KI-2009 for hosting the workshop at the KI-2009 conference. Finally, we would like to thank the Gesellschaft für Informatik, the TU Dortmund, and the FernUniversität in Hagen for supporting this workshop.

August 2009 Gabriele Kern-Isberner and Christoph Beierle

Contents

Relevance, Conditionals, and Defeasible Reasoning

James P. Delgrande

School of Computing Science,
Simon Fraser University,
Burnaby, B.C.,
Canada V5A 1S6.
jim@cs.sfu.ca

This talk discusses the notion of relevance as it pertains to statements of normality and defeasible reasoning in Artificial Intelligence. The role of relevant properties in defeasible reasoning is first covered, along with a discussion of how relevance has been addressed in different approaches to nonmonotonic reasoning. Following this, an approach for incorporating irrelevant properties in a conditional knowledge base is presented; and a notion of defeasible reasoning is introduced, based on this approach.

In the approach, a closure operation is defined, so that from a theory of defeasible conditionals an extension is obtained wherein irrelevant properties are satisfactorily incorporated. The approach is shown to have desirable formal properties and handles various commonsense examples appropriately. It is also shown that this approach can be captured in an iterative definition. In conclusion, it is argued that defeasible reasoning with commonsense normative conditionals is exactly captured via a sufficiently strong logic of defeasible conditionals together with this means of handling relevance.

Log_AB: An Algebraic Logic of Belief

Haythem O. Ismail

German University in Cairo
Department of Computer Science
haythem.ismail@guc.edu.eg

Abstract. $Log_A\mathbf{B}$ is a family of logics of belief. It holds a middle ground between the expressive, but prone to paradox, syntactical first-order theories and the often inconvenient, but safe, modal approaches. In this report, the syntax and semantics of $Log_A\mathbf{B}$ are presented. $Log_A\mathbf{B}$ is algebraic in the sense that it is a language of only terms; there is no notion of a formula, only proposition-denoting terms. The domain of propositions is taken to be a Boolean lattice, which renders classical truth conditions and definitions of consequence and validity theorems about $Log_A\mathbf{B}$ structures. $Log_A\mathbf{B}$ is shown to be sufficiently expressive to accommodate complex patterns of reasoning about belief while remaining paradox-free. A number of results are proved regarding paradoxical self-reference. They are shown to strengthen previous results, and to point to possible new approaches to circumventing paradoxes in syntactical theories of belief.

1 Introduction

Belief is usually viewed as a relation between a believing agent and a believed entity, typically a proposition or a sentence. Logics of belief come in two main flavors: the modal and the syntactical. Modal approaches [1–4, for instance] represent belief by a modal operator and employ some version of possible-worlds semantics. Syntactical theories [5–9, for instance] employ self-referential first-order languages, where belief is represented by a (typically) dyadic predicate of agents and sentences of the language. The semantics is standard Tarskian semantics, but complications arise due to the need to employ theories of arithmetic or string manipulation. On one hand, first-order logics are more expressive and more well-understood than their modal rivals. On the other hand, a result by Thomason [10] (following a similar result by Montague for the case of knowledge [11]) shows that, assuming some desirable properties of belief, first-order doxastic theories are paradoxical, whereas modal ones are not.

In this paper, I present $Log_A\mathbf{B}$, a family of algebraic logics of belief. $Log_A\mathbf{B}$ is algebraic in the sense that it only contains terms, algebraically constructed from function symbols. No sentences are included in a $Log_A\mathbf{B}$ language. Instead, there are terms of a distinguished syntactic type that are taken to denote propositions. The inclusion of propositions in the ontology, though non-standard, has been suggested by a few authors [6, 12, for instance]. I refer the reader to Shapiro's

article [12] for arguments in favor of adopting this approach in the representation of propositional attitudes in artificial intelligence. It turns out that, in addition to Shapiro's arguments, recognizing propositions as first-class inhabitants of our ontology has the additional benefit of avoiding the doxastic paradoxes referred to above. In particular, $Log_A\mathbf{B}$ holds a middle ground between modal and first-order syntactical theories of belief. On one hand, it is almost as expressive as the first-order theories; on the other hand, it is weak just enough to avoid the paradoxes to which those theories are susceptible.

Chalupsky and Shapiro [13] present a logic of belief, **SL**, based on Shapiro's proposal. $Log_A\mathbf{B}$ and **SL** differ in several important respects. Chiefly among these is that **SL** is fully-intensional; it adopts an excessively fine-grained representation of propositions [13, p. 168] and has no room for notions of truth, logical consequence, and validity.[1] $Log_A\mathbf{B}$ is much closer in spirit to standard extensional first-order theories. In the $Log_A\mathbf{B}$ ontology, propositions are structured in a Boolean lattice. This gives us, almost for free, all standard truth conditions, standard notions of consequence and validity, and an individuation of propositions that is neither too fine-grained, nor too coarse-grained, for a doxastic logic.[2] Moreover, Chalupsky and Shapiro are primarily concerned with simulative belief ascription, and include no mention of doxastic paradoxes of self-reference.

The paper is organized as follows. In Section 2, the syntax and semantics of $Log_A\mathbf{B}$ are presented. Section 3 shows how an account of truth may be associated with the otherwise truth-independent semantics of $Log_A\mathbf{B}$. Proof theory is briefly discussed in Section 4. Section 5 analyzes the notion of belief in terms of properties of $Log_A\mathbf{B}$ semantic structures. In Section 6, the expressivity of $Log_A\mathbf{B}$ is demonstrated by showing how notions of common and distributed belief (cf. [2]) may be accounted for. Results pertaining to paradoxes of self-reference are presented in Section 7: We (i) show that $Log_A\mathbf{B}$ is not susceptible to paradox, (ii) strengthen a previous result of Bolander's [9], and (iii) point out possible new approaches to circumventing paradox in syntactical theories. For completeness, an appendix includes relevant background on Boolean algebra.

2 Log_AB Languages

$Log_A\mathbf{B}$ is a class of languages that share a common core of logical symbols and differ in a signature of non-logical symbols. A $Log_A\mathbf{B}$ language is a set of terms partitioned into two base syntactic types, σ_P and σ_I. Intuitively, σ_P is the set of terms denoting propositions and σ_I is the set of terms denoting anything else. A distinguished subset σ_A of σ_I comprises agent-denoting terms. In more specialized uses of $Log_A\mathbf{B}$, the set σ_I may be further partitioned into more fine-grained syntactic types. For example, in a temporal setting, we can have a type for time-, state-, or event-denoting terms.

[1] Which is just fine for the purposes of Chalupsky and Shapiro in [13].

[2] The use of Boolean lattices may be seen as an application of the Boolean-valued models of set theory [14], or an extension of the mereology-based algebraic semantics of Link to the domain of propositions [15].

2.1 Syntax

As is customary in type-theoretical treatments, an alphabet of $Log_A\mathbf{B}$ is made up of a set of syncategorematic punctuation symbols and a set of denoting symbols each from a set σ of syntactic types. The set σ is the smallest set containing all of the following types.

1. σ_P.
2. σ_I.
3. $\tau_1 \longrightarrow \tau_2$, for $\tau_1 \in \{\sigma_P, \sigma_I\}$ and $\tau_2 \in \sigma$.

Intuitively, $\tau_1 \longrightarrow \tau_2$ is the syntactic type of function symbols that take a single argument of type σ_P or σ_I and produce a functional term of type τ_2. Given the restriction of the first argument of function symbols to base types, $Log_A\mathbf{B}$ is, in a sense, a first-order language.

A $Log_A\mathbf{B}$ alphabet is a union of four disjoint sets: $\Omega \cup \Xi \cup \Sigma \cup \Lambda$. The set Ω, the *signature* of the language, is a non-empty set of constant and function symbols. Each symbol in the signature has a designated syntactic type from σ and a designated adicity. (As usual, constants may be viewed as 0-adic function symbols.). Ω is what distinguishes one $Log_A\mathbf{B}$ language from another.

The set $\Xi = \{x_i, a_i, p_i\}_{i\in\mathbb{N}}$ is a countably infinite set of variables, where $x_i \in \sigma_I$, $a_i \in \sigma_A$, and $p_i \in \sigma_P$, for $i \in \mathbb{N}$. Σ is a set of syncategorematic symbols, including the comma, various matching pairs of brackets and parentheses, and the symbol $\forall$. The set Λ is the set of logical symbols of $Log_A\mathbf{B}$, defined as the union of the following sets.

1. $\{\neg\} \subseteq \sigma_P \longrightarrow \sigma_P$
2. $\{\wedge, \vee\} \subseteq \sigma_P \longrightarrow \sigma_P \longrightarrow \sigma_P$
3. $\{\mathbf{B}\} \subseteq \sigma_A \longrightarrow \sigma_P \longrightarrow \sigma_P$

A $Log_A\mathbf{B}$ language with signature Ω is denoted by L_Ω. It is the smallest set of terms formed according to the following rules, where t and t_i ($i \in \mathbb{N}$) are terms in L_Ω.

- $\Xi \subset L_\Omega$
- $c \in L_\Omega$, where $c \in \Omega$ is a constant symbol.
- $f(t_1, \ldots, t_n) \in L_\Omega$, where $f \in \Omega$ is of type $\tau_1 \longrightarrow \ldots \longrightarrow \tau_n \longrightarrow \tau$ and t_i is of type τ_i.
- $\neg t \in L_\Omega$, where $t \in \sigma_P$.
- $(t_1 \otimes t_2) \in L_\Omega$, where $\otimes \in \{\wedge, \vee\}$ and $t_1, t_2 \in \sigma_P$.
- $\forall x(t) \in L_\Omega$, where $x \in \Xi$ and $t \in \sigma_P$.
- $\mathbf{B}(t_1, t_2) \in L_\Omega$, where $t_1 \in \sigma_A$ and $t_2 \in \sigma_P$.

As usual, terms involving $\Rightarrow$, $\Leftrightarrow$, and $\exists$ may be introduced as abbreviations in the standard way.

2.2 Semantics

The basic ingredient of the $Log_A\mathbf{B}$ semantic apparatus is the notion of a $Log_A\mathbf{B}$ structure.

Definition 1 *A* $Log_A\mathbf{B}$ structure *is a triple* $\mathfrak{S} = \langle \mathcal{D}, \mathfrak{A}, \mathfrak{b} \rangle$*, where*

- $\mathcal{D}$*, the domain of discourse, is a set with two disjoint, non-empty, countable subsets* $\mathcal{P}$ *and* $\mathcal{A}$*.*
- $\mathfrak{A} = \langle \mathcal{P}, +, \cdot, -, \bot, \top \rangle$ *is a complete, non-degenerate Boolean algebra.*
- $\mathfrak{b} : \mathcal{A} \times \mathcal{P} \longrightarrow \mathcal{P}$*.*

Intuitively, the domain $\mathcal{D}$ is partitioned by a set of propositions $\mathcal{P}$, structured as a Boolean lattice, and a set of individuals $\overline{\mathcal{P}}$, among which at least one agent in the set $\mathcal{A}$ of agents.[3] These stand in correspondence to the syntactic sorts of $Log_A\mathbf{B}$. In what follows, we let $\mathcal{D}_{\sigma_P} = \mathcal{P}$, $\mathcal{D}_{\sigma_I} = \overline{\mathcal{P}}$, and $\mathcal{D}_{\sigma_A} = \mathcal{A}$.

Definition 2 *Let* L_Ω *be a* $Log_A\mathbf{B}$ *language. A* valuation $\mathcal{V}$ *of* L_Ω *is a pair* $\langle \mathfrak{S}, v_\Omega \rangle$*, where* $\mathfrak{S}$ *is a* $Log_A\mathbf{B}$ *structure; and* v_Ω *is a function that assigns to each constant of sort* τ *in* Ω *an element of* $\mathcal{D}_\tau$*, and to each n-adic* ($n \geq 1$) *function symbol* $f \in \Omega$ *of sort* $\tau_1 \longrightarrow \ldots \longrightarrow \tau_n \longrightarrow \tau$ *an n-adic function* $v_\Omega(f) : \bigtimes_{i=1}^{n} \mathcal{D}_{\tau_i} \longrightarrow \mathcal{D}_\tau$*.*

Definition 3 *Let* L_Ω *be a* $Log_A\mathbf{B}$ *language and let* $\mathcal{V}$ *be a valuation of* L_Ω*. For a variable assignment* $v_\Xi : \Xi \longrightarrow \mathcal{D}$*, where, for every* $i \in \mathbb{N}$*,* $v_\Xi(x_i) \in \overline{\mathcal{P}}$*,* $v_\Xi(a_i) \in \mathcal{A}$*, and* $v_\Xi(p_i) \in \mathcal{P}$*, an* interpretation *of the terms of* L_Ω *is given by a function* $[\![\cdot]\!]^{\mathcal{V},v_\Xi}$*:*

- $[\![x]\!]^{\mathcal{V},v_\Xi} = v_\Xi(x)$*, for* $x \in \Xi$
- $[\![c]\!]^{\mathcal{V},v_\Xi} = v_\Omega(c)$*, for a constant* $c \in \Omega$
- $[\![f(t_1, \ldots, t_n)]\!]^{\mathcal{V},v_\Xi} = v_\Omega(f)([\![t_1]\!]^{\mathcal{V},v_\Xi}, \ldots, [\![t_n]\!]^{\mathcal{V},v_\Xi})$*, for an n-adic* ($n \geq 1$) *function symbol* $f \in \Omega$
- $[\![(t_1 \wedge t_2)]\!]^{\mathcal{V},v_\Xi} = [\![t_1]\!]^{\mathcal{V},v_\Xi} \cdot [\![t_2]\!]^{\mathcal{V},v_\Xi}$
- $[\![(t_1 \vee t_2)]\!]^{\mathcal{V},v_\Xi} = [\![t_1]\!]^{\mathcal{V},v_\Xi} + [\![t_2]\!]^{\mathcal{V},v_\Xi}$
- $[\![\neg t]\!]^{\mathcal{V},v_\Xi} = -[\![t]\!]^{\mathcal{V},v_\Xi}$
- $[\![\forall x(t)]\!]^{\mathcal{V},v_\Xi} = \prod_{a \in \mathcal{D}_\tau} [\![t]\!]^{\mathcal{V},v_\Xi[a/x]}$*, where* x *is of sort* τ*,* $v_\Xi[a/x](x) = a$*, and* $v_\Xi[a/x](y) = v_\Xi[a/x](y)$ *for every* $y \neq x$
- $[\![\mathbf{B}(t_1, t_2)]\!]^{\mathcal{V},v_\Xi} = \mathfrak{b}([\![t_1]\!]^{\mathcal{V},v_\Xi}, [\![t_2]\!]^{\mathcal{V},v_\Xi})$

In $Log_A\mathbf{B}$, logical consequence is defined in pure algebraic terms without alluding to the notion of truth. This is achieved using the natural partial order $\leq$ associated with $\mathfrak{A}$. (See the appendix for details.)

[3] I will have nothing much to say about the contentious issue of what propositions really are. I take propositions at least to be abstract particulars that are distinct from sentences or terms denoting them.

Definition 4 *Let L_Ω be a $Log_A\mathbf{B}$ language. For every $\phi \in \sigma_P$ and $\Gamma \subseteq \sigma_P$, ϕ is a logical consequence of Γ, denoted $\Gamma \models \phi$, if, for every L_Ω valuation $\mathcal{V}$ and $Log_A\mathbf{B}$ variable assignment v_Ξ, $\prod_{\gamma\in\Gamma} [\![\gamma]\!]^{\mathcal{V},v_\Xi} \leq [\![\phi]\!]^{\mathcal{V},v_\Xi}$.*

By the above definition, and the algebraic properties of $\mathfrak{S}$, we can easily verify the validity of the following typical examples of logical consequence:

$$\{\phi \wedge \psi\} \models \phi, \{\phi\} \models \phi \vee \psi, \{\phi \Rightarrow \psi, \phi\} \models \psi, \{\bot\} \models \phi, \{\phi\} \models \top$$

In the appendix it is shown that $\models$ has the distinctive properties of classical Tarskian logical consequence.

Proposition 1 *Let L_Ω be a $Log_A\mathbf{B}$ language with $\phi \in L_\Omega$ and $\Gamma, \Delta \subseteq L_\Omega$.*

1. *If $\phi \in \Gamma$, then $\Gamma \models \phi$.*
2. *If $\Gamma \models \phi$ and $\Gamma \subseteq \Delta$, then $\Delta \models \phi$.*
3. *If $\Gamma \models \psi$ and $\Gamma \cup \{\psi\} \models \phi$, then $\Gamma \models \phi$.*

Definition 5 *Let L_Ω be a $Log_A\mathbf{B}$ language. For every $\phi, \psi \in \sigma_P$, ϕ is logically equivalent to ψ, denoted $\phi \equiv \psi$, if, for every L_Ω valuation $\mathcal{V}$ and $Log_A\mathbf{B}$ variable assignment v_Ξ, $[\![\phi]\!]^{\mathcal{V},v_\Xi} = [\![\psi]\!]^{\mathcal{V},v_\Xi}$. ϕ is logically valid if $[\![\phi]\!]^{\mathcal{V},v_\Xi} = \top$, for every L_Ω valuation $\mathcal{V}$ and $Log_A\mathbf{B}$ variable assignment v_Ξ.*

Again, all the standard logical equivalences are valid in our system as a direct corollary to the properties of Boolean algebras. Thus, for example, $\phi \wedge \psi$ and $\psi \wedge \phi$ are two different terms denoting the same proposition, and, hence, are logically equivalent.

3 Truth

As is clear from the previous section, the semantics of $Log_A\mathbf{B}$ has no place for a notion of truth. While we can happily accommodate the standard semantic relations of consequence and equivalence and the property of logical validity, our semantic apparatus has nothing to say about truth. But perhaps this is fine; for truth in the world and the language we use to describe that world and to carry out reasoning about it are not necessarily dependent.

However, it seems that we should at least provide truth conditions for σ_P terms of a $Log_A\mathbf{B}$ language. In standard Tarskian semantics, truth conditions of propositions are part of the definition of the interpretation of the language (Definition 3, in our case). We, however, seem to need more. What we need is what I shall call a *world structure*—a structure describing exactly which propositions in a $Log_A\mathbf{B}$ structure are true in the world.

Definition 6 *For every $Log_A\mathbf{B}$ structure $\mathfrak{S} = \langle \mathcal{D}, \mathfrak{A}, \mathfrak{b} \rangle$, a bivalent world structure $\mathfrak{W}_\mathbf{2}(\mathfrak{S})$ is a countably-complete ultrafilter of $\mathfrak{A}$.*[4]

[4] For the definition of ultrafilters, check the appendix. In future work, n-valent world structures are to be considered.

Intuitively, the world structure $\mathfrak{W}_{\mathbf{2}}(\mathfrak{S})$ comprises the set of propositions that are true. Members of the corresponding maximal ideal $\overline{\mathfrak{W}_{\mathbf{2}}(\mathfrak{S})}$ are the false propositions.

In what follows, a bivalent model of a $Log_A\mathbf{B}$ language L_Ω is a triple $\mathcal{M}_{\mathbf{2}} = \langle \mathfrak{W}_{\mathbf{2}}(\mathfrak{S}), \mathcal{V}, v_\Xi \rangle$, where $\mathfrak{W}_{\mathbf{2}}(\mathfrak{S})$ is a bivalent world structure, $\mathcal{V} = \langle \mathfrak{S}, v_\Omega \rangle$ is an L_Ω valuation, and v_Ξ is a $Log_A\mathbf{B}$ variable assignment.

Definition 7 *Let L_Ω be a $Log_A\mathbf{B}$ language. A σ_P-term $\phi \in L_\Omega$ is* true *in a bivalent model $\mathcal{M}_{\mathbf{2}} = \langle \mathfrak{W}_{\mathbf{2}}(\mathfrak{S}), \mathcal{V}, v_\Xi \rangle$, denoted $True_{\mathcal{M}_{\mathbf{2}}}(\phi)$, if $[\![\phi]\!]^{\mathcal{V}, v_\Xi} \in \mathfrak{W}_{\mathbf{2}}(\mathfrak{S})$. Otherwise, ϕ is* false *in $\mathcal{M}_{\mathbf{2}}$, denoted $False_{\mathcal{M}_{\mathbf{2}}}(\phi)$.*

With a bivalent world structure, a $Log_A\mathbf{B}$ logic satisfies the laws of bivalence, excluded-middle, and non-contradiction.

Proposition 2 *Let L_Ω be a $Log_A\mathbf{B}$ language with a bivalent model $\mathcal{M}_{\mathbf{2}}$. For every $\phi \in \sigma_P$ the following is true.*

1. *$True_{\mathcal{M}_{\mathbf{2}}}(\phi)$ or $False_{\mathcal{M}_{\mathbf{2}}}(\phi)$*
2. *$True_{\mathcal{M}_{\mathbf{2}}}(\phi)$ or $True_{\mathcal{M}_{\mathbf{2}}}(\neg\phi)$*
3. *It is not the case that both $True_{\mathcal{M}_{\mathbf{2}}}(\phi)$ and $True_{\mathcal{M}_{\mathbf{2}}}(\neg\phi)$*

The classical truth conditions for compound propositions follow from the above definition.

Proposition 3 *Let L_Ω be a $Log_A\mathbf{B}$ language with a bivalent model $\mathcal{M}_{\mathbf{2}} = \langle \mathfrak{W}_{\mathbf{2}}(\mathfrak{S}), \mathcal{V}, v_\Xi \rangle$ and let $\phi, \psi \in \sigma_P$ and $x \in \tau$.*

- *$True_{\mathcal{M}_{\mathbf{2}}}(\neg\phi)$ if and only if $False_{\mathcal{M}_{\mathbf{2}}}(\phi)$.*
- *$True_{\mathcal{M}_{\mathbf{2}}}(\phi \wedge \psi)$ if and only if $True_{\mathcal{M}_{\mathbf{2}}}(\phi)$ and $True_{\mathcal{M}_{\mathbf{2}}}(\psi)$.*
- *$True_{\mathcal{M}_{\mathbf{2}}}(\phi \vee \psi)$ if and only if $True_{\mathcal{M}_{\mathbf{2}}}(\phi)$ or $True_{\mathcal{M}_{\mathbf{2}}}(\psi)$.*
- *$True_{\mathcal{M}_{\mathbf{2}}}(\forall x(\phi))$ if and only if $True_{\mathcal{M}^b_{\mathbf{2}}}(\phi)$, for all $b \in \mathcal{D}_\tau$, where $\mathcal{M}^b_{\mathbf{2}}(\phi)$ is identical to $\mathcal{M}_{\mathbf{2}}(\phi)$ with v_Ξ replaced by $v_\Xi[b/x]$.*

Typically, Proposition 3 is given as the definition of truth conditions. In our system, the definition is given by membership in some ultrafilter of the underlying Boolean algebra of propositions. Now, one might suspect that there is something unsatisfying about the current state of affairs. For, whereas Proposition 3 provides the classical truth conditions for compound propositions, it is silent about atomic ones. The only thing that we have to say about the truth conditions of atomic propositions is said in Definition 7. But, according to this definition, an atomic propositional term such as $Dog(fido)$ is true if the proposition it denotes is true—something that is determined by fiat. This does not seem to explain much if compared to the classical assignment of truth based on Fido's membership in the extension of the predicate *Dog*.

Nevertheless, the membership test for atomic propositions features, albeit in a slightly different guise, in our semantics. Given a structure $\mathfrak{S}$, the semantics of a $Log_A\mathbf{B}$ symbol like *Dog* is a function from individuals to propositions that those individuals are dogs. However, a model $\mathcal{M}$ (including a world structure

for $\mathfrak{S}$) gives rise to a derived function from individuals to truth values, roughly $True_{\mathcal{M}} \circ v_{\Omega}(Dog)$. But this is clearly the characteristic function of the classical set of dogs provided by a Tarskian model. Thus, whatever notion of meaning is provided by classical semantics is also inherent in our algebraic semantics. In addition, we seem to provide a level of meaning (the proposition), independent of a world structure, which is not explicitly available in classical theories.

4 Proof Theory

Our proof theory assumes a (possibly empty) finite knowledge base $\mathbb{K} \subset \sigma_P$ and an inference canon. I will take the inference canon to be a set of Fitch-style natural deduction rules of inference. Such rules come in two forms:

$$\frac{\Gamma}{\phi} \text{ and } \frac{\Gamma, \Delta}{\phi}$$

where $\phi \in \sigma_P$, Γ is a finite subset of σ_P-terms, and Δ is a finite set of items of the form $\Gamma_i \vdash \psi_i$, $\Gamma_i \cup \{\psi_i\} \subset \sigma_P$. As usual, the first form is interpreted as follows: If $\Gamma \subseteq \mathbb{K}$, then ϕ may be added to $\mathbb{K}$. For the second form, if $\Gamma \subseteq \mathbb{K}$ and ψ_i is derivable by the rules of inference with Γ_i as the knowledge base, for every $\Gamma_i \vdash \psi_i \in \Delta$, then ϕ may be added to $\mathbb{K}$. The notion of derivation is given the standard definition in terms of a finite sequence of justified σ_p-terms that ends with the derived expression. I will have nothing more to say about the proof theory here, but any system of Fitch-style natural deduction that is sound and complete for first-order logic will also be sound and complete for $Log_A\mathbf{B}$. I will also not commit myself to any particular set of rules or axiom schema for belief at this point, but Sections 5 and 7 present a thorough discussion of what the possibilities are.

5 Properties of Belief

Given the $Log_A\mathbf{B}$ semantics presented so far, our notion of belief, beside being a relation between agents and propositions, is otherwise totally unconstrained. Although flexibility is a virtue, we may still want our notion of belief to have certain reasonable properties. In what follows, I list some of these.

Definition 8 *Let $\mathfrak{S} = \langle \mathcal{D}, \mathfrak{A}, \mathfrak{b} \rangle$ be a $Log_A\mathbf{B}$ structure.*

1. $\mathfrak{S}$ *is* injective *if $\mathfrak{b}$ is injective.*
2. $\mathfrak{S}$ *is* non-trivial *if $Range(\mathfrak{b}) \cap \mathfrak{W}_{\mathbf{2}}(\mathfrak{S}) \not\subseteq \{\top\}$, for every bivalent world structure $\mathfrak{W}_{\mathbf{2}}(\mathfrak{S})$.*
3. $\mathfrak{S}$ *is* meet-distributive *if $\mathfrak{b}(a, p \cdot q) = \mathfrak{b}(a, p) \cdot \mathfrak{b}(a, q)$, for every $p, q \in \mathcal{P}$, $a \in \mathcal{A}$.*
4. $\mathfrak{S}$ *is* join-distributive *if $\mathfrak{b}(a, p) + \mathfrak{b}(a, q) = \mathfrak{b}(a, p + q)$, for every $p, q \in \mathcal{P}$, $a \in \mathcal{A}$.*
5. $\mathfrak{S}$ *is* consistent *if for every $(a, p) \in \mathcal{A} \times \mathcal{P}$, $\mathfrak{b}(a, -p) \leq -\mathfrak{b}(a, p)$.*

6. *A pair* $(a,p) \in \mathcal{A} \times \mathcal{P}$ *is an* autoepistemic pair *if* $-\mathfrak{b}(a,p) \leq \mathfrak{b}(a,-p)$. $\mathfrak{S}$ *is* autoepistemic *if every* $(a,p) \in \mathcal{A} \times \mathcal{P}$ *is an autoepistemic pair.*
7. $\mathfrak{S}$ *is* positively- (negatively-) introspective *if, for every* $(a,p) \in \mathcal{A} \times \mathcal{P}$, $q \leq \mathfrak{b}(a,q)$, *for* $q = \mathfrak{b}(a,p)$ *(respectively,* $-\mathfrak{b}(a,p)$*).*
8. $\mathfrak{S}$ *is* faithful *if, for every* $(a,p) \in \mathcal{A} \times \mathcal{P}$, $\mathfrak{b}(a,\mathfrak{b}(a,p)) \leq \mathfrak{b}(a,p)$.[5]

For injection, the intuition is that the proposition that a believes p is different from the proposition that b believes q, unless $a = b$ and $p = q$. This property is, in general, absent for other $\mathcal{D}$-valued functions. For example, the father of John may be identical to the father of Mary, and the proposition that John is a sibling of Mary may be identical to the proposition that Mary is a sibling of John. Note, however, that full injection may lead to awkward structures in the presence of other properties. For example, if $\mathfrak{S}$ is both positively-introspective and faithful, then $\mathfrak{b}(a,p) = \mathfrak{b}(a,\mathfrak{b}(a,p))$, for any $(a,p) \in \mathcal{A} \times \mathcal{P}$. Injection implies that $p = \mathfrak{b}(a,p)$, which trivializes the whole notion of belief. Careful definition of $\mathfrak{b}$ is, thus, recommended to avoid such anomalies.[6]

A trivial structure is one for which some world structure only admits either non-believing agents (practically, automata), or agents that believe only what they are bound to believe. While it might not seem reasonable to assume that $\top \in \mathit{Range}(\mathfrak{b})$, we do not rule out this possibility. For example, someone might argue that $\mathfrak{b}(a,\top) = \top$, mirroring the rule of necessity in modal doxastic logic: It is logically valid to believe what is logically valid. Also, sometimes the condition of conceit, $\mathfrak{b}(a,-\mathfrak{b}(a,p)+p) = \top$, is advisable (cf. [1, 10]).[7] Nevertheless, this will not be very useful in the absence of non-trivial beliefs.

Meet-distributivity is a strong condition that implies logical omniscience (cf. Observation 1 below). In general, this property should not be tolerated if we would like to account for realistic agents. The same applies to join-distributivity. Typically, only one direction of join-distributivity is desirable, namely $\mathfrak{b}(p) + \mathfrak{b}(q) \leq \mathfrak{b}(p+q)$. Unfortunately, this direction is *equivalent* to logical omniscience. Better than meet- and join-distributivity, a syntactic version involving $\wedge$ and $\vee$ instead of $\cdot$ and $+$ is preferred.[8]

The above properties of belief are not independent. The following observation lists some dependencies that will turn out to be important in Section 7.

Observation 1 *Let* $\mathfrak{S} = \langle \mathcal{D}, \mathfrak{A}, \mathfrak{b} \rangle$ *be a* $\mathit{Log}_A\mathbf{B}$ *structure.*

1. *If* $\mathfrak{S}$ *is meet-distributive (join-distributive) and* $p \leq q$, *for* $p,q \in \mathcal{P}$, *then* $\mathfrak{b}(a,p) \leq \mathfrak{b}(a,q)$, *for any* $a \in \mathcal{A}$.
2. *If* $\mathfrak{S}$ *is autoepistemic, consistent, and meet-distributive, then it is join-distributive.*

[5] The label "faithful" is inspired by [16].

[6] In their fully-intensional logic, Chalupsky and Shapiro [13] avoid these anomalies by refraining from making any assumptions about belief, including positive introspection and faithfulness.

[7] The label "conceit" is due to [17].

[8] In that case, our **B** will behave similar to Levesque's modality of explicit belief [3, p. 201].

3. *If $\mathfrak{S}$ is consistent and negatively-introspective, then it is faithful.*
4. *If $\mathfrak{S}$ is autoepistemic, consistent, positively-introspective, and meet-distributive then it is negatively-introspective.*
5. *If $\mathfrak{S}$ is autoepistemic and faithful, then it is negatively-introspective.*

6 Expressivity

How expressive is $Log_A\mathbf{B}$? As expected, it is more expressive than modal theories of belief. In particular, we can quantify over beliefs, which allows us, for example, to limit properties like introspection to any intensionally characterized set of agents. To demonstrate the expressivity of $Log_A\mathbf{B}$, consider the following example.

Example 1. In [2], the authors describe a proposition to be common knowledge for a group of agents if all agents in the group know it, all agents in the group know that all agents in the group know it, etc. In the modal framework adopted in [2], the authors had to introduce a new modal operator (actually, a class thereof, one operator for each possible group) to capture this notion of common knowledge. In $Log_A\mathbf{B}$, we can make use of our ability to quantify over propositions in order to represent the corresponding notion of *common belief*. If our group of agents is intensionally characterized by G, we first introduce a non-logical function symbol $\mathbf{B}^*_G$, which is akin to the reflexive transitive closure of $\mathbf{B}$ (viewed relationally):

$$\forall p\ [\mathbf{B}^*_G(p,p)]$$

$$\forall a,p\ [\mathbf{B}^*_G(p,\mathbf{B}(a,p)) \Leftrightarrow G(a)]$$

$$\forall p,q,r\ [\mathbf{B}^*_G(p,q) \wedge \mathbf{B}^*_G(q,r) \Rightarrow \mathbf{B}^*_G(p,r)]$$

Common belief (**CB**) can be defined as follows.

$$\mathbf{CB}_G(p) =_{\text{def}} \forall a\ [G(a) \Rightarrow \forall q\ [\mathbf{B}^*_G(p,q) \Rightarrow \mathbf{B}(a,q)]]$$

In [2], the related notion of distributed knowledge characterizes those propositions that are implied by the collective knowledge of a group. Again, the authors introduced a new modal operator to model distributed knowledge. In $Log_A\mathbf{B}$, we need a non-logical function symbol capturing the collective beliefs of the group:

$$\mathbf{B}^{\cup}_G(p) \Leftrightarrow \exists a[G(a) \wedge \mathbf{B}(a,p)]\ \vee\ \exists q,r[\mathbf{B}^{\cup}_G(q) \wedge \mathbf{B}^{\cup}_G(r) \wedge (p \Leftrightarrow (q \wedge r))]$$

We may now define distributive belief as follows.

$$\mathbf{DB}_G(p) =_{\text{def}} \exists q\ [\mathbf{B}^{\cup}_G(q) \wedge (q \Rightarrow p)]$$

□

Though more expressive than modal theories, $Log_A\mathbf{B}$ is, in a certain sense, less expressive than *syntactical* first-order theories. In particular, $Log_A\mathbf{B}$ languages are not self referential.

7 Self-Reference

Results of Montague [11] and Thomason [10] show that a first-order treatment of epistemic modalities (respectively, knowledge and belief) yield inconsistent systems. Of course, it is a particular mix of assumptions about the modalities that gives rise to inconsistencies. The inconsistencies appear as paradoxes of self-reference, akin to the famous Liar paradox. How does our system fair in this regard? Interestingly, our system is immune to such paradoxes for the same reason why it is not a classical first-order logic. Let me explain. First-order doxastic theories are *syntactical*, they include a dyadic belief predicate (akin to our functional $\mathbf{B}$) whose first argument is an agent-denoting term and whose second argument is a term that denotes a *formula* of the same language. It is this ability of the language to refer to its own syntactic structures that makes such theories syntactical. However, such ability does not come for free; there are, in general, two methods to achieve syntacticity. The first is to equip the language with an axiomatization of arithmetic and denote formulas of the language by their Gödel numbers [18, 9, for instance]. The second is to provide the language with systematic means to manipulate strings, together with devices for substitution, quotation, and un-quotation [7, for instance].

In syntactical theories, we can have formulas that refer to themselves. For example, a formula $P(\ulcorner 123 \urcorner)$ may have as its Gödel number the very same 123, encoded by the string $\ulcorner 123 \urcorner$. In fact, the diagonalization lemma (see [19, for example]) states that, in a syntactical first-order theory, there is a formula ϕ such that $\phi \Leftrightarrow p(\ulcorner \phi \urcorner)$ is a theorem, for any (possibly complex) monadic predicate p. Note that $\phi \Leftrightarrow p(\ulcorner \phi \urcorner)$ is a *theorem*—we have no way of avoiding it—not just a sentence generated by the grammar of the language. It is this result that leads to doxastic paradoxes, when p is $\lambda x.\neg B(\alpha, x)$ and B is the belief predicate.

As demonstrated by several authors [20, 8, 21], it is the syntacticity of a system that is the catalyst for paradox, not whether it is first-order or modal. Interestingly, $Log_A\mathbf{B}$, which has all the advantages that a first-order doxastic theory has over a modal one (see Section 6), is not syntactical. There is no way for a $Log_A\mathbf{B}$ language to refer to its own terms. In particular, no σ_P term can refer to itself, since, *tout court*, the proper-substring relation is irreflexive. Granted, we can write expressions such as $\phi \Leftrightarrow \neg\mathbf{B}(\alpha, \phi)$ (or even $\phi = \neg\mathbf{B}(\alpha, \phi)$); we can have such expressions in our knowledge base; and, with a certain notorious suite of assumptions on $\mathbf{B}$, we shall get a contradiction. But this inconsistency is an inconsistency of the knowledge base, not a natural product of our proof theory: paradoxical, self-referential expressions are *not* theorems of our logic.

Now, the immunity of $Log_A\mathbf{B}$ to paradox is clearly rooted in its relative expressive weakness, compared to syntactical theories, when it comes to representing its own syntax. However, syntactical theories have almost exclusively been employed to account for the propositional attitudes, which a $Log_A\mathbf{B}$ approach seems to effectively accommodate. Perlis [7, 8] argues that languages with self-reference are essential for commonsense reasoning in general. I might have something to say about this, but that is a story for another day.

Notwithstanding $Log_A\mathbf{B}$'s immunity to paradox, we can certainly construct $Log_A\mathbf{B}$ structures in which, for some $p \in \mathcal{P}$, $p = -\mathfrak{b}(a,p)$. Such structures will be incompatible with some of the properties in Definition 8. Syntactically, this means that (knowledge-base) inconsistency looms given $\phi \Leftrightarrow \mathbf{B}(\alpha,\phi)$ and specific axiomatizations of belief. In what follows, we prove several results, indicating exactly when that happens. We start with the following two useful lemmas.[9]

Lemma 1. *If $\mathfrak{S}$ is a consistent, positively-introspective $Log_A\mathbf{B}$ structure, then $p = -\mathfrak{b}(a,p)$ implies $p = \top$ for every $(a,p) \in \mathcal{A} \times \mathcal{P}$.*
Proof. Suppose that $p = -\mathfrak{b}(a,p)$. By consistency, $\mathfrak{b}(a,-\mathfrak{b}(a,p)) = \mathfrak{b}(a,p) \leq -\mathfrak{b}(a,\mathfrak{b}(a,p))$. On the other hand, by positive introspection, $\mathfrak{b}(a,p) \leq \mathfrak{b}(a,\mathfrak{b}(a,p))$. It follows that, $\mathfrak{b}(a,p) \leq -\mathfrak{b}(a,\mathfrak{b}(a,p)) \cdot \mathfrak{b}(a,\mathfrak{b}(a,p)) = \bot$. Thus, $\mathfrak{b}(a,p) = \bot$ and, hence, $p = \top$ (by the definition of $\leq$ and B7.1 in the appendix). □

Lemma 2. *If $\mathfrak{S}$ is a negatively-introspective $Log_A\mathbf{B}$ structure, then $p = -\mathfrak{b}(a,p)$ implies $p = \bot$ for every $(a,p) \in \mathcal{A} \times \mathcal{P}$.*
Proof. Assume that $p = -\mathfrak{b}(a,p)$. It follows from negative introspection that $-\mathfrak{b}(a,p) \leq \mathfrak{b}(a,-\mathfrak{b}(a,p))$. Then $-\mathfrak{b}(a,p) \leq \mathfrak{b}(a,p)$, and, consequently, $-\mathfrak{b}(a,p) = p = \bot$ (by the definition of $\leq$ and B5.2 in the appendix). □

Our first theorem is a variant of Theorem 4.7 in [9, p. 76], where the inconsistency result of Thomason [10] is regenerated by trading conceit ($\mathfrak{b}(a,-\mathfrak{b}(a,p)+p) = \top$) for the more subjective negative introspection.

Theorem 1. *If $\mathfrak{S}$ is a consistent, positively-, and negatively-introspective $Log_A\mathbf{B}$ structure, then there is no $(a,p) \in \mathcal{A} \times \mathcal{P}$ such that $p = -\mathfrak{b}(a,p)$.*
Proof. Assume $(a,p) \in \mathcal{A} \times \mathcal{P}$ such that $p = -\mathfrak{b}(a,p)$. By Lemma 1, $p = \top$. By Lemma 2, $p = \bot$. Consequently, $\bot = \top$, which is impossible since the algebra $\mathfrak{A}$ is non-degenerate. □

The following theorem shows that positive introspection is not responsible, after all, for the inconsistency.

Theorem 2. *If $\mathfrak{S}$ is a consistent, negatively-introspective, and meet-distributive $Log_A\mathbf{B}$ structure, then there is no $(a,p) \in \mathcal{A} \times \mathcal{P}$ such that $p = -\mathfrak{b}(a,p)$.*
Proof. Assume $(a,p) \in \mathcal{A} \times \mathcal{P}$ such that $p = -\mathfrak{b}(a,p)$. By negative introspection and Lemma 2, $p = \bot$. Consequently, $\mathfrak{b}(a,p) = \mathfrak{b}(a,\bot) = \top$. Now, let $q \in \mathcal{P}$ be an arbitrary proposition. Since $\bot = q \cdot -q$, it follows that $\mathfrak{b}(a,q \cdot -q) = \top$. By meet-distributivity, $\mathfrak{b}(a,q \cdot -q) = \mathfrak{b}(a,q) \cdot \mathfrak{b}(a,-q)$. By consistency, $\mathfrak{b}(a,q \cdot -q) \leq \mathfrak{b}(a,q) \cdot -\mathfrak{b}(a,q) = \bot$, which is impossible since $\mathfrak{A}$ is non-degenerate. □

Even though we require $\mathfrak{S}$ to be meet-distributive, the direction of meet-distributivity actually used in the proof ($\mathfrak{b}(a,p \cdot q) \leq \mathfrak{b}(a,p) \cdot \mathfrak{b}(a,q)$) is provably equivalent to logical omniscience ($p \leq q$ implies $\mathfrak{b}(a,p) \leq \mathfrak{b}(a,q)$). Logical omniscience is already a property of the systems of both Thomason [10] and Bolander

[9] Strictly speaking, standard results pertain to sentences, not propositions. But see [22] for a discussion of how such results extend to propositions.

[9]. Hence, the above result is a strengthening of Bolander's in that it shows that negative-introspection, consistency, and omniscience are sufficient—without positive introspection—to regenerate Thomason's paradox. In addition, the result also shows that negative introspection is problematic enough to induce an inconsistency if it replaces Thomason's conceit and positive introspection.

But, while negative introspection is rightly incriminated by the above result, it certainly is not necessary for the inconsistency. The following theorem shows that replacing negative-introspection with the less controversial property of faithfulness gives rise to inconsistency when a non-pervasive version of autoepistemology is enforced.[10]

Theorem 3. *If $\mathfrak{S}$ is a consistent, positively-introspective, and faithful $Log_A\mathbf{B}$ structure, then there is no autoepistemic pair (a,p) such that $p = -\mathfrak{b}(a,p)$.*
Proof. Assume there is an autoepistemic pair $(a,p) \in \mathcal{A} \times \mathcal{P}$ such that $p = -\mathfrak{b}(a,p)$. By Lemma 1, $p = -\mathfrak{b}(a,p) = \top$. Since (a,p) is an autoepistemic pair, $-\mathfrak{b}(a,p) \leq \mathfrak{b}(a,-p) = \mathfrak{b}(a,\mathfrak{b}(a,p))$. By faithfulness and transitivity of $\leq$, $\top = -\mathfrak{b}(a,p) \leq \mathfrak{b}(a,p) = \bot$. Consequently, $\bot = \top$, which is impossible since the algebra $\mathfrak{A}$ is non-degenerate. □

In the spirit of [23], we present the following non-theorem, demonstrating the necessity of autoepistemology for the above result.

Non-Theorem 1 *If $\mathfrak{S}$ is a consistent, positively-introspective, and faithful $Log_A\mathbf{B}$ structure, then there is no $(a,p) \in \mathcal{A} \times \mathcal{P}$ such that $p = -\mathfrak{b}(a,p)$.*
Counterexample 1. Let $\mathfrak{S}_{TWO} = \langle \mathcal{D}_2, \mathfrak{A}_2, \mathfrak{b}_2 \rangle$. Take $\mathcal{D}_2 = \{a, \bot, \top\}$, $\mathfrak{A}_2 = \langle \{\bot,\top\}, +, \cdot, -, \bot, \top \rangle$, and $Range(\mathfrak{b}_2) = \{\bot\}$. $\mathfrak{S}_{TWO}$ is trivially consistent. In addition, it is both positively-introspective and faithful, since $\mathfrak{b}_2(a,p) = \mathfrak{b}_2(a,\mathfrak{b}_2(a,p)) = \bot$ for $p \in \mathcal{P}$. In this structure, $\top = -\bot = -\mathfrak{b}_2(a,\top)$.

Our first counterexample, though falsifies the non-theorem, constructs a rather trivial structure. Our second example is more general.
Counterexample 2. Consider two disjoint sets $P = \{p_1, \ldots, p_n\}$ and $Q = \{q_1, \ldots, q_n\}$ of propositions. Let $\mathfrak{A}_P = \langle P', +, \cdot, -, \bot, \top \rangle$ and $\mathfrak{A}_Q = \langle Q', +, \cdot, -, \bot, \top \rangle$ be the Boolean algebras generated by P and Q, respectively, and let $\mathfrak{A}_\cup = \langle (P \cup Q)', +, \cdot, -, \bot, \top \rangle$ be the Boolean algebra generated by $P \cup Q$. Take $\mathfrak{i}$ to be an isomorphism from $\mathfrak{A}_P$ to $\mathfrak{A}_Q$, where $\mathfrak{i}(p_i) = q_i$ for $p_i \in P$. Now, define $\mathfrak{S}_{ISO} = \langle \{a\} \cup (P \cup Q)', \mathfrak{A}_\cup, \mathfrak{b}_\mathfrak{i} \rangle$, where $\mathfrak{b}_\mathfrak{i}$ is defined as follows.

$$\mathfrak{b}_\mathfrak{i}(a,x) = \begin{cases} \bot & \text{if } x \in \{\bot,\top\} \\ \mathfrak{i}(x) & \text{if } x \in P' \setminus \{\bot,\top\} \\ x & \text{if } x \in Q' \setminus \{\bot,\top\} \\ z & \text{if } x = y \odot z, \text{ where } \odot \in \{\cdot,+\}, y \in P' \setminus \{\bot,\top\}, \\ & \text{and } z \in Q' \setminus \{\bot,\top\} \end{cases}$$

It could be shown, by induction on the structure of x, that $\mathfrak{b}_\mathfrak{i}$ is well-defined. We now show that it satisfies the conditions stated in the non-theorem. First,

[10] Faithfulness is a theorem of Thomason's system [10]. Hence, the following result is, in a sense, a strengthening of Thomason's.

consider consistency. The case of $\bot$ and $\top$ is similar to Counterexample 1. For $p \in P' \setminus \{\bot, \top\}$, $\mathfrak{b}_{\mathfrak{i}}(a, -p) = \mathfrak{i}(-p) = -\mathfrak{i}(p) = -\mathfrak{b}_{\mathfrak{i}}(a, p)$. For $q \in Q' \setminus \{\bot, \top\}$, $\mathfrak{b}_{\mathfrak{i}}(a, -q) = -q = -\mathfrak{b}_{\mathfrak{i}}(a, q)$. For $x = y \cdot z$, with $y \in P' \setminus \{\bot, \top\}$ and $z \in Q' \setminus \{\bot, \top\}$, $\mathfrak{b}_{\mathfrak{i}}(a, -x) = \mathfrak{b}_{\mathfrak{i}}(a, -y + -z) = -z = -\mathfrak{b}_{\mathfrak{i}}(a, y \cdot z)$. Similarly for $x = y + z$. Note that, not only is $\mathfrak{S}$ consistent, but it is also the case that every pair $(a, x) \in \mathcal{A} \times (P \cup Q)' \setminus \{\bot, \top\}$ is autoepistemic.

Similar to Counterexample 1, $\mathfrak{b}_{\mathfrak{i}}(a, x) = \mathfrak{b}_{\mathfrak{i}}(a, \mathfrak{b}_{\mathfrak{i}}(a, x))$, for $x \in \{\bot, \top\}$. Otherwise, $\mathfrak{b}_{\mathfrak{i}}(a, x) \in Q' \setminus \{\top\}$ and is, hence, identical to $\mathfrak{b}_{\mathfrak{i}}(a, \mathfrak{b}_{\mathfrak{i}}(a, x))$. Thus, $\mathfrak{S}$ is both positively-introspective and faithful. Finally, similar to Counterexample 1, note that $\top = -\bot = -\mathfrak{b}_{\mathfrak{i}}(a, \top)$. □

In the $Log_A\mathbf{B}$ structure $\mathfrak{S}_{ISO}$ of Counterexample 2, all propositions, except the paradoxical one, are autoepistemic and satisfy the negative introspection schema. In addition, $\mathfrak{S}_{ISO}$ is *almost* meet-distributive (and join-distributive), which means that it almost satisfies logical omniscience (cf. Observation 1). Thus, not only have we shown that consistency, positive-introspection, and faithfulness are tolerant to the $p = -\mathfrak{b}(a, p)$ possibility, but we have also shown that the tolerance persists even in the presence of a high degree of logical omniscience. We are pretty close to Thomason's system.

It should be clear that $\mathfrak{S}_{ISO}$ could be varied along different dimensions. We may allow multiple agents, where instead of the single set Q, we have a family of sets indexed by $\mathcal{A}$. Omniscience may also be avoided by constructing the isomorphism differently. For example, instead of standing in 1-1 correspondence to the set P, Q can be defined to correspond 1-1 to the elements of the *algebra* generated by P. We may also construct a structure in which we are more conservative about which propositions are autoepistemic. One way to achieve this is to change the definition of $\mathfrak{b}$ such that $\mathfrak{b}(a, -p) = -\mathfrak{b}(a, p) \cdot AE(p)$, where $AE(p) = \top$ only for some $p \in \mathcal{P}$ (those that are intuitively autoepistemic). Note that this definition of $\mathfrak{b}$ maintains the property of consistency.

Theorem 3 and Non-theorem 1 tell us the following: For consistent, positively-introspective, and faithful (and almost omniscient) structures, a pair (a, p) will satisfy $p = -\mathfrak{b}(a, p)$ if and only if it is not autoepistemic. What is interesting here is that the offensive property—autoepistemology—is one that naturally applies only to select propositions (and agents) by fiat. For example, whereas my having a brother is plausibly autoepistemic, my first-grade teacher's being asleep right now is clearly not. Thus, if pressed, we may deem a proposition p, such that $p = -\mathfrak{b}(a, p)$, non-autoepistemic. Now, while it might be easy to semantically implement this decree (and to perhaps philosophically justify it), it is not immediately clear how we can syntactically enforce it. But the results obtained in [23, 18, 9] give us some hope. Based on purely syntactic properties, we may be able to quarantine some σ_P terms which are believed to give rise to paradoxical self-reference. Unlike [23, 18, 9], where the recommendation is to suspend the application of *all* doxastic schema on the quarantined expressions, we only need to make sure that we do not label any of them as autoepistemic. We, thus, get the full force of rational belief (for example, consistency, positive introspection, faithfulness), and the limited application of negative introspection to the au-

toepistemic agent-proposition pairs (cf. Observation 1). The exact ramifications of this result for syntactical theories is to be explored in future work.

8 Conclusions

By admitting propositions as first-class individuals in our ontology, we achieve two things: (i) the expressivity and semantic simplicity of first-order doxastic theories and (ii) the consistency and syntactic simplicity of rival modal theories. I hope I have managed to convince the reader of the above claim through the presentation of $Log_A\mathbf{B}$. No paradoxical self-referential propositional term is a theorem of $Log_A\mathbf{B}$, but results of Thomason's and Bolander's feature as conditions of incompatibility of certain properties of $Log_A\mathbf{B}$'s semantics structures. This leads to one direction of future work: How may the results presented here (in particular those pertaining to the interrelations among autoepistemology, faithfulness, conceit, and negative introspection) be applied to syntactical theories in order to avoid paradox, while minimally sacrificing the pervasive adoption of desirable properties of belief?

Other directions for future research include developing similar algebraic accounts for other propositional attitudes, notably knowledge (within a $Log_A\mathbf{K}$ framework). Also, as pointed out earlier, non-bivalent world structures of $Log_A\mathbf{B}$ may be studied. In particular, a trivalent world structure, corresponding to a three-valued logic, may be defined as a filter (as opposed to an ultrafilter) of the underlying Boolean algebra. Similarly, a quad-valent world structure could be defined as a filter-ideal pair. Connections of such systems to existing many-valued logics (for example, [24, 25]) are then to be systematically studied.

References

1. Hintikka, J.: Knowledge and Belief: An Introduction to the Logic of the Two Notions. Cornell University Press, Ithaca, New York (1962)
2. Fagin, R., Halpern, J., Moses, Y., Vardi, M.: Reasoning about Knowledge. The MIT Press, Cambridge, Massachusetts (1995)
3. Levesque, H.: A logic of implicit and explicit belief. In: Proceedings of the Fourth National Conference on Artificial Intelligence (AAAI-84), Palo Alto, California, Morgan Kaufmann (1984) 198–202
4. Kaplan, A., Schubert, L.: A computational model of belief. Artificial Intelligence **120** (2000) 119–160
5. Kaplan, D.: Quantifying in. Synthese **19** (1968) 178–214
6. Bealer, G.: Theories of properties, relations, and propositions. The Journal of Philosophy **76**(11) (1979) 634–648
7. Perlis, D.: Languages with self-refernce I: Foundations. Artificial Intelligence **25** (1985) 301–322
8. Perlis, D.: Languages with self-reference II: Knowledge, belief, and modality. Artificial Intelligence **34**(2) (1988) 179–212
9. Bolander, T.: Logical Theories for Agent Introspection. PhD thesis, Department of Informatics and Mathematical Modelling, Technical University of Denmark (2003)

10. Thomason, R.: A note on syntactical treatments of modality. Synthese **44**(3) (1980) 391–395
11. Montague, R.: Syntactical treatments of modality, with corollaries on reflexion principles and finite axiomatizability. Acta Philosophica Fennica **16** (1963) 153–167
12. Shapiro, S.C.: Belief spaces as sets of propositions. Journal of Experimental and Theoretical Artificial Intelligence **5** (1993) 225–235
13. Chalupsky, H., Shapiro, S.: SL: A subjective, intensional logic of belief. In: Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society, Hillsdale, New Jersey, Lawrence Erlbaum (1994) 165–170
14. Bell, J.: Set Theory: Boolean-Valued Models and Independence Proofs. 3rd edn. Oxford University Press (2005)
15. Link, G.: Algebraic Semantics in Language and Philosophy. CSLI Publications, Stanford, CA (1998)
16. Konolige, K.: A computational theory of belief introspection. In: Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85), Los Angeles, California (1985) 502–508
17. Smullyan, R.: Logicians who reason about themselves. In [27] 342–352
18. Morreau, M., Kraus, S.: Syntactical treatments of propositional attitudes. Artificial Intelligence **106** (1998) 161–177
19. Boolos, G., Burgess, J., Jeffrey, R.: Computability and Logic. 5th edn. Cambridge University Press (2007)
20. Asher, N., Kamp, H.: The knower's paradox and representational theories of attitudes. In [27] 131–148
21. Grim, P.: Operators in the paradox of the knower. Synthese **94**(3) (1993) 409–428
22. Mills, E.: A simple solution to the liar. Philosophical Studies **89**(2/3) (1998) 197–212
23. des Rivières, J., Levesque, H.: The consistency of syntactical treatments of knowledge. In [27] 115–130
24. Kleene, S.C.: On notation for ordinal numbers. The Journal of Symbolic Logic **3**(4) (1938) 150–155
25. Belnap, N.: A useful four-valued logic: How a computer should think. In Anderson, A., Belnap, N., Dunn, M., eds.: Entailment. Volume II. Princeton University Press, Princeton, NJ (1992) 506–541
26. Burris, S., Sankappanavar, H.P.: A Course in Universal Algebra. Springer-Verlag (1982)
27. Halpern, J., ed.: Proceedings of the First International Conference on Theoretical Aspects of Reasoning about Knowledge (TARK-86), Monterey, California, Morgan Kaufmann (1986)

Appendix: Boolean Algebra

For the sake of completeness, I hereafter present some basic results about Boolean algebra that are relevant to the development of $Log_A\mathbf{B}$. The presentation is based primarily on [26]. All proofs are omitted for limitations of space; the interested reader may consult [26] or any standard text on the topic.

A Boolean algebra is a sextuple $\mathfrak{B} = \langle B, +, \cdot, -, \bot, \top\rangle$ where B is a nonempty set and $\{\bot, \top\} \subseteq B$. B is closed under the two binary operators $+$ and $\cdot$ and the unary operator $-$. The operators satisfy the following conditions.

B1.1: $a + b = b + a$ (Commutativity)
B1.2: $a \cdot b = b \cdot a$
B2.1: $a + (b + c) = (a + b) + c$ (Associativity)
B2.2: $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
B3.1: $a + (a \cdot b) = a$ (Absorption)
B3.2: $a \cdot (a + b) = a$
B4.1: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ (Distribution)
B5.1: $a + -a = \top$ (Complements)
B5.2: $a \cdot -a = \bot$

The following properties of Boolean algebras immediately follow.

B4.2: $a + (b \cdot c) = (a + b) \cdot (a + c)$
B6.1: $a \cdot a = a$
B6.2: $a + a = a$
B7.1: $a \cdot \bot = \bot$
B7.2: $a + \top = \top$
B8: $a \cdot \top = a + \bot = a$
B9: $-(-a) = a$
B10.1: $-(a \cdot b) = (-a) + (-b)$
B10.2: $-(a + b) = (-a) \cdot (-b)$

A Boolean algebra $\mathfrak{B} = \langle B, +, \cdot, -, \bot, \top \rangle$ is *complete* if, for every $A \subseteq B$, $\sum_{a \in A} a \in B$ and $\prod_{a \in A} a \in B$. $\mathfrak{B}$ is *degenerate* if $\bot = \top$, otherwise, it is *non-degenerate*. Elements of B are partially-ordered by the relation $\leq$, where $a \leq b$ if and only if $a \cdot b = a$. By B3.1 and B3.2, it follows that $a \leq b$ if and only if $a + b = b$.

A *filter* of $\mathfrak{B}$ is a subset F of B such that

F1. $\top \in F$
F2. $a, b \in F$ implies $a \cdot b \in F$
F3. $a \in F$ and $a \leq b$ imply $b \in F$

F is an *ultrafilter* of $\mathfrak{B}$ if it is maximal with respect to not including $\bot$. The following properties of ultrafilters follow from the definitions.

F4. For every $a \in B$, exactly one of a and $-a$ belong to F.
F5. For every $a, b \in B$, $a + b \in F$ if and only if $a \in F$ or $b \in F$.

Moreover, F is *countably-complete* if it satisfies

F6. For every $A \subseteq F$, if A is countable, then $\prod_{a \in A} a \in F$.

To illustrate the relevance of the above properties of Boolean algebras to $Log_A\mathbf{B}$, I present proofs for Propositions 1 and 3. (Proposition 2 follows immediately from F4.)

Proof of Proposition 1.

1. Suppose that $\phi \in \Gamma$. Then, for every L_Ω valuation $\mathcal{V}$ and $Log_A\mathbf{B}$ variable assignment v_Ξ, $\prod_{\gamma\in\Gamma} [\![\gamma]\!]^{\mathcal{V},v_\Xi} = (\prod_{\phi\neq\gamma\in\Gamma} [\![\gamma]\!]^{\mathcal{V},v_\Xi}) \cdot [\![\phi]\!]^{\mathcal{V},v_\Xi}$. By B2.2 and B6.1,

$$(\prod_{\gamma\in\Gamma} [\![\gamma]\!]^{\mathcal{V},v_\Xi}) \cdot [\![\phi]\!]^{\mathcal{V},v_\Xi} = (\prod_{\phi\neq\gamma\in\Gamma} [\![\gamma]\!]^{\mathcal{V},v_\Xi}) \cdot ([\![\phi]\!]^{\mathcal{V},v_\Xi} \cdot [\![\phi]\!]^{\mathcal{V},v_\Xi}) = \prod_{\gamma\in\Gamma} [\![\gamma]\!]^{\mathcal{V},v_\Xi}$$

   Hence, $\Gamma \models \phi$.
2. Suppose that $\Gamma \models \phi$ and $\Gamma \subseteq \Delta$. Thus, there is a set Γ' such that $\Delta = \Gamma \cup \Gamma'$ and $\Gamma \cap \Gamma' = \varnothing$. By B1.2 and B2.2,

$$(\prod_{\delta\in\Delta} [\![\delta]\!]^{\mathcal{V},v_\Xi}) \cdot [\![\phi]\!]^{\mathcal{V},v_\Xi} = (\prod_{\gamma'\in\Gamma'} [\![\gamma']\!]^{\mathcal{V},v_\Xi}) \cdot [(\prod_{\gamma\in\Gamma} [\![\gamma]\!]^{\mathcal{V},v_\Xi}) \cdot [\![\phi]\!]^{\mathcal{V},v_\Xi}]$$

   Since $\Gamma \models \phi$, it follows that

$$(\prod_{\delta\in\Delta} [\![\delta]\!]^{\mathcal{V},v_\Xi}) \cdot [\![\phi]\!]^{\mathcal{V},v_\Xi} = (\prod_{\gamma'\in\Gamma'} [\![\gamma']\!]^{\mathcal{V},v_\Xi}) \cdot (\prod_{\gamma\in\Gamma} [\![\gamma]\!]^{\mathcal{V},v_\Xi}) = \prod_{\delta\in\Delta} [\![\delta]\!]^{\mathcal{V},v_\Xi}$$

   Hence, $\Delta \models \phi$.
3. Suppose $\Gamma \models \psi$ and $\Gamma \cup \{\psi\} \models \phi$. By definition of $\models$,

$$(\prod_{\gamma\in\Gamma} [\![\gamma]\!]^{\mathcal{V},v_\Xi}) \cdot [\![\psi]\!]^{\mathcal{V},v_\Xi} \leq [\![\phi]\!]^{\mathcal{V},v_\Xi}$$

   But, since $\Gamma \models \psi$,

$$\prod_{\gamma\in\Gamma} [\![\gamma]\!]^{\mathcal{V},v_\Xi} = (\prod_{\gamma\in\Gamma} [\![\gamma]\!]^{\mathcal{V},v_\Xi}) \cdot [\![\psi]\!]^{\mathcal{V},v_\Xi} \leq [\![\phi]\!]^{\mathcal{V},v_\Xi}$$

   Hence, $\Gamma \models \phi$. □

Proof of Proposition 3.

1. $True_{\mathcal{M}_\mathbf{2}}(\neg\phi)$ iff $-[\![\phi]\!]^{\mathcal{V},v_\Xi} \in \mathfrak{W}_\mathbf{2}(\mathfrak{S})$ iff $[\![\phi]\!]^{\mathcal{V},v_\Xi} \notin \mathfrak{W}_\mathbf{2}(\mathfrak{S})$ (by F4) iff $False_{\mathcal{M}_\mathbf{2}}(\phi)$.
2. $True_{\mathcal{M}_\mathbf{2}}(\phi \wedge \psi)$ iff $[\![\phi]\!]^{\mathcal{V},v_\Xi} \cdot [\![\psi]\!]^{\mathcal{V},v_\Xi} \in \mathfrak{W}_\mathbf{2}(\mathfrak{S})$ iff $[\![\phi]\!]^{\mathcal{V},v_\Xi} \in \mathfrak{W}_\mathbf{2}(\mathfrak{S})$ and $[\![\psi]\!]^{\mathcal{V},v_\Xi} \in \mathfrak{W}_\mathbf{2}(\mathfrak{S})$ (by F2 and F3) iff $True_{\mathcal{M}_\mathbf{2}}(\phi)$ and $True_{\mathcal{M}_\mathbf{2}}(\psi)$.
3. $True_{\mathcal{M}_\mathbf{2}}(\phi \vee \psi)$ iff $[\![\phi]\!]^{\mathcal{V},v_\Xi} + [\![\psi]\!]^{\mathcal{V},v_\Xi} \in \mathfrak{W}_\mathbf{2}(\mathfrak{S})$ iff $[\![\phi]\!]^{\mathcal{V},v_\Xi} \in \mathfrak{W}_\mathbf{2}(\mathfrak{S})$ or $[\![\psi]\!]^{\mathcal{V},v_\Xi} \in \mathfrak{W}_\mathbf{2}(\mathfrak{S})$ (by F5) iff $True_{\mathcal{M}_\mathbf{2}}(\phi)$ or $True_{\mathcal{M}_\mathbf{2}}(\psi)$.
4. Similar to the case of $\wedge$, using F6 instead of F2. □

First-Order Probabilistic Conditional Logic
Introduction and Representation

Jens Fisseler

Faculty of Mathematics and Computer Science, FernUniversität in Hagen, Germany

Abstract. Combining probability and first-order logic has been the subject of intensive research during the last ten years. This paper introduces first-order probabilistic conditional logic, a first-order extension of a propositional probabilistic logic representation formalism, which allows for the adequate representation of probabilistic *if-then*-rules. As the representation of the models of first-order probabilistic conditional logic requires solving a complex entropy-optimization problem, we devise syntactic conditions for the simplification of this optimization problem.

1 Introduction

Knowledge representation and reasoning is one of the main research topics of artificial intelligence. As most real-world knowledge is uncertain rather than certain, knowledge representation formalisms should be able to deal with this uncertainty. One approach to represent and process uncertain knowledge is probability theory [1,2], which, with the introduction of *probabilistic graphical models*, has seen increasing research interest during the last two decades. Markov and Bayesian networks are two well-known classes of probabilistic graphical models [3,4], but only allow the representation of propositional probabilistic knowledge.

However, as many real-world knowledge representation tasks require the ability to represent uncertain knowledge about a varying number of objects and their (uncertain) relationships, several approaches for combining probabilistic graphical models and some subset of first-order logic have been developed, see Chapter 10 of [5] as well as [6] for an overview. The best known of these formalisms are *probabilistic relational models* (*PRMs*), *Bayesian logic programs* (*BLPs*), and *Markov logic networks* (*MLNs*), see corresponding Chapters in [5]. Although their models are defined by "templates", specified by using some subset of first-order logic, for inference, these formalisms work at a propositional level: PRMs and BLPs induce a Bayesian network, whereas MLNs induce a Markov network. The formulas of PRMs and BLPs are parametrized with conditional probability functions, whereas a MLN consists of weighted formulas. Therefore, MLNs are not as easily comprehensible, and it is also difficult to specify them by hand, e.g. as background knowledge for learning. However, formalisms which are mapped to Bayesian networks have difficulties representing circular dependencies.

In this paper, we introduce an alternative approach for combining a subset of first-order logic and probabilistic models which allows to specify models via probabilistic conditionals. Because these probabilistic conditionals do not specify a

unique model, we use the principle of maximum entropy as a model selection criterion, which yields a convex optimization problem, with one optimization variable for each ground instance of a first-order probabilistic conditional. As solving this optimization problem is computationally infeasible for all but the smallest sets of ground instances, we develop syntactic conditions which ensure that some or all ground instances of the same conditional "share" the same entropy-optimal parameter value, which greatly simplifies the entropy-optimization problem.

The following section motivates the need for first-order probabilistic conditional logic by pointing out some drawback of other formalisms. Section 3 introduces syntax and semantics of our novel formalism, and Section 4 presents some example knowledge bases. Section 5 discusses the representation of maximum-entropy models of first-order probabilistic conditional logic, presenting syntactical conditions which ensure "sharing" of the same entropy-optimal parameter value between ground instances of the same first-order probabilistic conditional. Section 6 concludes.

2 Motivating First-Order Probabilistic Conditional Logic

Assume we want to formalize the following uncertain knowledge about having resp. catching a common cold:

$$\begin{array}{lll} \text{R1:} & & \mathit{common\text{-}cold}(U)\ [0.01] \\ \text{R2:} & \textbf{if} & \mathit{susceptible}(U) \\ & \textbf{then} & \mathit{common\text{-}cold}(U)\ [0.1] \\ \text{R3:} & \textbf{if} & \mathit{contact}(U,V) \\ & \textbf{and} & \mathit{common\text{-}cold}(V) \\ & \textbf{then} & \mathit{common\text{-}cold}(U)\ [0.6] \end{array} \tag{1}$$

The uncertain *if-then*-rule R1 states that one normally does not have a common cold, i.e. only with a diminutive probability of 0.01. Rule R2 denotes that a person catches a common cold with probability 0.1 if this person is susceptible to it, and rule R3 represents the knowledge that person U, which is in contact with another person V which has the common cold, also gets a common cold with probability 0.6.

Formalisms based on directed graphical models cannot represent this uncertain knowledge, as rule R3 depicts a circular dependency, which cannot be modeled with Bayesian networks. Hence, one could try to encode this knowledge base with Markov logic, representing the *if-then*-rules using material implication, which is denoted here by "$\leftarrow$":

$$\begin{array}{lll} \text{R1}_{\text{ML}} : \mathit{common\text{-}cold}(U). & & [w_1] \\ \text{R2}_{\text{ML}} : \mathit{common\text{-}cold}(U) & \leftarrow \mathit{susceptible}(U). & [w_2] \\ \text{R3}_{\text{ML}} : \mathit{common\text{-}cold}(U) & \leftarrow \mathit{contact}(U,V) \\ & \quad \wedge \mathit{common\text{-}cold}(V). & [w_3] \end{array} \tag{2}$$

Having defined the Markov logic formulas, the next step is to specify their weights w_1–w_3. For Markov logic, there is no direct method for computing the formulas' weights from prescribed probabilities, but [7] gives an intuitive interpretation for the weight of a formula F when it is the *only* formula of a MLN: its weight w can be viewed as the log-odds between a world where F is true and a world where it is false, other things being equal.

Though Equation (2) contains three formulas, one might still choose their weights based on this intuition, which yields $w_1 := \log \frac{1}{100}$, $w_2 := \log \frac{1}{10}$, and $w_3 := \log \frac{6}{10}$. However, as the ground instances of the formulas $\mathrm{R1_{ML}}$–$\mathrm{R3_{ML}}$ share some of their ground atoms, they influence each other, and therefore this naïve approach for specifying the weights is not suitable. For example, with constant symbols $C = \{a, b\}$, the probability (cf. Equation (4) in [7]) of person a having a common cold is $p(\mathit{common\text{-}cold}(a)) = 0.066416$, which is significantly higher than the desired probability 0.01, which has been used to specify w_1. On the other hand, the probability of a ground instance of $\mathrm{R3_{ML}}$, e.g. $p(\mathit{common\text{-}cold}(a) \mid \mathit{contact}(a, b) \wedge \mathit{common\text{-}cold}(b))$, is 0.031946, which is much lower than the prescribed probability of 0.6. Therefore, the intuitive interpretation of the weight of a formula is not appropriate for computing the weights from prescribed formula probabilities.

As there is no direct method for calculating the weights of Markov logic formulas from prescribed probabilities, a pragmatic way to obtain these parameters is learning them from data. While this approach is more or less unproblematic when learning MLNs from data, for knowledge representation, only prescribed probabilities for the given formulas are available. Therefore, an appropriate data set must be generated first, within which the given formulas should hold with their desired probabilities. From this data set, the formulas' weights can subsequently be learned. For complex knowledge bases, the generation of an appropriate data set is all but straightforward, as the formulas influence or interact with each other, i.e. changing some parts of the data set to get the desired probability of one formula right might change the probability of another formula.

Though feasible, due to lack of space, we cannot go into details of how to generate an appropriate data set which represents the formulas $\mathrm{R1_{ML}}$–$\mathrm{R3_{ML}}$ with their prescribed probabilities. Instead, we point out another drawback of Markov logic networks: their inability to model conditional probabilities. The weights attached to the formulas of a Markov logic network—despite their non-probabilistic interpretation—correspond to marginal probabilities for their associated formulas. Even material implications, which have been used to model the conditional statements represented by the uncertain *if-then*-rules R1–R3, can only be quantified by weights corresponding to marginal probabilities. However, uncertain conditional statements can only be adequately quantified by conditional probabilities [8,9], which cannot be represented by Markov logic networks.

In summary, we have shown that several well known approaches for combining first-order logic and probabilistic graphical models have problems representing a knowledge base consisting of probabilistic *if-then*-rules. Formalisms using Bayesian networks cannot represent circular dependencies, whereas Markov logic

networks are not easily specified by hand. Furthermore, MLNs cannot represent conditional probabilities, which are the only adequate quantification of probabilistic conditionals. The following sections present and discuss a formalism addressing these issues.

3 First-Order Probabilistic Conditional Logic

This section introduces the syntax and semantics of *first-order probabilistic conditional logic (*FO-PCL*)*, which can be viewed as a restricted first-order extension of (propositional) probabilistic conditional logic, see [10]. Please note that a similar formalism has been previously presented in [11].

3.1 Syntax

Let $\Sigma = (S, \mathcal{D}, \mathit{Pred})$ be a function-free, many-sorted signature[1], consisting of a non-empty set S of *sorts*, an S-indexed family of pairwise disjoint, non-empty sets of constant symbols $\mathcal{D} = \{D^{(s)}\}_{s \in S}$, and a non-empty, S^*-indexed set *Pred* of predicate symbols. As the sets of constant symbols are pairwise disjoint, we can assume $\mathcal{D} = \bigcup_{s \in S} D^{(s)}$. Furthermore, *Pred* can also be assumed to be a single set instead of a family of sets. Every predicate symbol $p_{s_1 \ldots s_n} \in \mathit{Pred}$ has an associated index $s_1 \ldots s_n \in S^*$, and $p_{s_1 \ldots s_n}$ is also written as $p/\langle s_1, \ldots, s_n \rangle$, where n denotes the *arity* of p. The empty sequence of sorts is denoted by ε, and nullary predicate symbols (which correspond to propositions) are written as p/ε. Every many-sorted signature Σ contains some *predefined predicate symbols*, which consist of $\top/\varepsilon$ and $\bot/\varepsilon$, as well as $=^{(s)}/\langle s, s \rangle$, for every sort $s \in S$.

For every sort $s \in S$, $V^{(s)}$ denotes an enumerable set of *variables of sort* s, and the set of all variables is defined as $\mathcal{V} := \bigcup_{s \in S} V^{(s)}$. The set $\mathit{Term}_\Sigma^{(s)}$ of *terms of sort* s is defined as $\mathit{Term}_\Sigma^{(s)} := V^{(s)} \cup D^{(s)}$, and the set of all terms is denoted by $\mathit{Term}_\Sigma := \bigcup_{s \in S} \mathit{Term}_\Sigma^{(s)} = \mathcal{V} \cup \mathcal{D}$.

Analogous to other formalisms combining a subset of first-order logic and probabilistic graphical models, FO-PCL knowledge bases are propositionalized by substituting the variables of each FO-PCL "template"—probabilistic conditionals—with all admissible combinations of constant symbols. But contrary to the other approaches, straightforward instantiation of FO-PCL conditionals might easily results in inconsistencies. This can be illustrated on rule R3 depicted in Equation (1):

$$\begin{array}{lll} \text{R3:} & \textbf{if} & \mathit{contact}(U, V) \\ & \textbf{and} & \mathit{common\text{-}cold}(V) \\ & \textbf{then} & \mathit{common\text{-}cold}(U)\ [0.6] \end{array}$$

If the variables U and V would be substituted by the same constant symbol, the resulting ground instance of R3 would state that a person which has the

[1] We use many-sorted signatures to allow for some computational savings when instantiating an FO-PCL knowledge base.

common cold with certainty, i.e. with probability 1.0, and which is in contact with itself, has the common cold with a probability of 0.6 only, which is obviously inconsistent. Therefore, to avoid inconsistencies by restricting the set of admissible substitutions, first-order probabilistic conditional logic uses so-called *constraint formulas*, which are used to represent syntactical equalities and disequalities between terms, see [12,13] for a general introduction. Hence FO-PCL uses two different kinds of formulas, *logical formulas* and *constraint formulas*.

The set of *logical formulas* is the least set that contains *atomic logical formulas* of the form $p(t_1, \ldots, t_n)$, with $p/\langle s_1 \ldots s_n \rangle \in \mathit{Pred} \setminus \{=^{(s)}/\langle s, s \rangle \mid s \in S\}$ and $t_1 \in \mathit{Term}_{\Sigma}^{(s_1)}$, $\ldots$, $t_n \in \mathit{Term}_{\Sigma}^{(s_n)}$, nullary predicate symbols, and which is closed under the logical connectives $\neg$, $\vee$, and $\wedge$.

The set of *constraint formulas* is the least set that contains *equality constraint formulas* of the form $=^{(s)}(t_1, t_2)$, with $t_1, t_2 \in \mathit{Term}_{\Sigma}^{(s)}$, $s \in S$, $\top/\varepsilon$ and $\bot/\varepsilon$, and which is closed under the logical connectives $\neg$, $\vee$, and $\wedge$. A negated equality constraint formula $\neg(t_1 =^{(s)} t_2)$ is also written as $t_1 \neq^{(s)} t_2$, and is called a *disequality constraint formula*.

A *first-order probabilistic conditional* is an expression $\langle (\phi \mid \psi)[\xi], C \rangle$, consisting of two logical formulas ψ (the *premise*) and ϕ (the *conclusion*), *conditional probability* $\xi \in [0, 1]$, and constraint formula C. The variables of the constraint formula must be a subset of the variables of premise and conclusion: $\mathit{vars}(C) \subseteq (\mathit{vars}(\phi) \cup \mathit{vars}(\psi))$, where $\mathit{vars}(\cdot)$ denotes the set of variables of a logical or constraint formula.

Before defining the semantics of first-order probabilistic conditional logic, we present an example to further illustrate the syntactical concepts introduced in this section.

Example 1 ("Common Cold").

The "Common Cold" example models the uncertain knowledge pertaining to the causes for catching a common cold. This is an FO-PCL knowledge base for the uncertain *if-then*-rules presented in Equation (1):

$$\begin{aligned} &\text{CC1: } \langle (\mathit{common\text{-}cold}(U))[0.01], \top \rangle \\ &\text{CC2: } \langle (\mathit{common\text{-}cold}(U) \mid \mathit{susceptible}(U))[0.1], \top \rangle \\ &\text{CC3: } \langle (\mathit{common\text{-}cold}(U) \mid \mathit{contact}(U, V) \\ &\qquad\qquad \wedge \mathit{common\text{-}cold}(V))[0.6], U \neq V \rangle \end{aligned} \tag{3}$$

The signature $\Sigma_{\mathrm{CC}} = (S_{\mathrm{CC}}, \mathcal{D}_{\mathrm{CC}}, \mathit{Pred}_{\mathrm{CC}})$ of this example contains only one sort, *Person*. Its set $\mathcal{D}_{\mathrm{CC}} = D^{(\mathit{Person})}$ of constant symbols may contain any number of persons, and its set $\mathit{Pred}_{\mathrm{CC}}$ of predicate symbols consists of $\mathit{common\text{-}cold}/\langle \mathit{Person} \rangle$, $\mathit{susceptible}/\langle \mathit{Person} \rangle$ and $\mathit{contact}/\langle \mathit{Person}, \mathit{Person} \rangle$, as well as the predefined predicate symbols $=/\langle \mathit{Person}, \mathit{Person} \rangle$, $\top/\varepsilon$, and $\bot/\varepsilon$.

The conditionals CC1–CC3 should be self-explanatory, except perhaps for the constraint "$U \neq V$" of CC3, which simply states that no person is in contact with itself. Within the context of CC3, this means that no person can infect itself with the common cold, which avoids inconsistencies, see the discussion above.

3.2 Semantics

First-order probabilistic conditional logic uses a *possible worlds semantics* [14], analogous to most other formalisms for combining first-order logic and probabilistic graphical models. Therefore, we must define the set of possible worlds induced by a given set $\mathcal{R} = \{R_1, \ldots, R_m\}$ of probabilistic conditionals R_k, $1 \leq k \leq m$, and we must specify a probability function of these possible worlds, resp. their corresponding random variables.

Analogous to the templates used by PRMs, BLPs and MLNs, the FO-PCL conditionals $\mathcal{R}$ are instantiated, which is done by substituting the variables of each conditional $R_k \in \mathcal{R}$ by all admissible ground substitutions. A *ground substitution* is a sort-respecting mapping $\theta : \mathcal{V} \to \mathit{Term}_\Sigma$, which is the identity for all but a finite set of variables, its *domain* $\mathit{dom}(\theta)$, and which maps the variables in $\mathit{dom}(\theta)$ onto constant symbols in $\mathcal{D}$. For any set of variables $V \subset \mathcal{V}$, $\Theta_\Sigma(V)$ denotes the *set of all ground substitutions for* V. The *application* of a ground substitution $\theta \in \Theta_\Sigma(V)$, $V \subset \mathcal{V}$, on a logical or constraint formula ρ is denoted by $\theta(\rho)$. If $\mathit{vars}(\rho) \subseteq \mathit{dom}(\theta)$, $\theta(\rho)$ is ground.

The set of admissible ground substitutions for a first-order probabilistic conditional $R = \langle (\phi \,|\, \psi)[\xi], C \rangle$ is restricted by its constraint formula C. A ground substitution $\theta \in \Theta_\Sigma(\mathit{vars}(\phi) \cup \mathit{vars}(\psi))$ is *admissible*, if $[\![\theta(C)]\!] = \mathit{true}$, where the constraint evaluation of equality constraint formulas is defined as

$$[\![t_1 =^{(s)} t_2]\!] := \begin{cases} \mathit{true} & \text{iff } t_1 \text{ and } t_2 \text{ depict the same term in } \mathit{Term}_\Sigma^{(s)},\ s \in S, \\ \mathit{false} & \text{otherwise,} \end{cases}$$

which is canonically extended to composite constraint formulas. The set of all admissible ground substitutions for the conditional $R = \langle (\phi \,|\, \psi)[\xi], C \rangle$ is defined as $\mathit{Sol}(C, \mathit{vars}(\phi) \cup \mathit{vars}(\psi)) := \{\theta \in \Theta_\Sigma(\mathit{vars}(\phi) \cup \mathit{vars}(\psi)) \mid [\![\theta(C)]\!] = \mathit{true}\}$, and the set of ground instances of R is denoted by

$$\mathit{gnd}(R) := \{\, \langle (\theta(\phi) \,|\, \theta(\psi))[\xi], \top \rangle \mid \theta \in \mathit{Sol}(C, \mathit{vars}(\phi) \cup \mathit{vars}(\psi)) \}\,, \tag{4}$$

which is canonically extended to the set $\mathit{gnd}(\mathcal{R})$ of ground instances of a set $\mathcal{R}$ of FO-PCL conditionals.

The set $\mathit{gnd}(R)$ of ground instances of a first-order probabilistic conditional $R = \langle (\phi \,|\, \psi)[\xi], C \rangle$ induces a set $\boldsymbol{X}(R)$ of binary random variables, which corresponds to the set of all syntactically different ground atoms of the ground instances in $\mathit{gnd}(R)$, and which is also the *Herbrand base* $\mathcal{H}(R)$ of R. Any subset $M \subseteq \mathcal{H}(R)$ is a *Herbrand interpretation*, and corresponds to a unique *configuration* $\boldsymbol{x} \in \mathcal{X}(R) := \{\mathit{true}, \mathit{false}\}^{|\mathcal{H}(R)|}$ of the random variables $\boldsymbol{X}(R)$. The configurations resp. Herbrand interpretations correspond to the *possible worlds*, and an FO-PCL *interpretation* is simply a joint probability function $p_{\boldsymbol{X}(R)}$ of $\boldsymbol{X}(R)$. Based on these definitions, the probability of a ground logical formula ρ with respect to a joint probability function $p_{\boldsymbol{X}(R)}$ is defined as

$$p_{\boldsymbol{X}(R)}(\rho) := \sum_{\substack{\boldsymbol{x} \in \mathcal{X}(R), \\ \boldsymbol{x} \models \rho}} p_{\boldsymbol{X}(R)}(\boldsymbol{x}), \tag{5}$$

where $\models$ denotes the usual *Herbrand satisfiability*. An FO-PCL interpretation $p_{\boldsymbol{X}(R)}$ for $R = \langle (\phi \,|\, \psi)[\xi], C \rangle$ is an FO-PCL *model for* R, iff

$$\forall \theta \in \mathit{Sol}(C, \mathit{vars}(\phi) \cup \mathit{vars}(\psi)) : p_{\boldsymbol{X}(R)}\left(\theta(\phi) \wedge \theta(\psi)\right) = \xi \cdot p_{\boldsymbol{X}(R)}\left(\theta(\psi)\right). \quad (6)$$

This can be equivalently represented by

$$\forall g_R \in \mathit{gnd}(R) : \sum_{\boldsymbol{x} \in \mathcal{X}(R)} f_{g_R}(\boldsymbol{x}) p_{\boldsymbol{X}(R)}(\boldsymbol{x}) = \xi, \quad (7)$$

where, for every ground instance $g_R = \langle (\theta_{g_R}(\phi) \,|\, \theta_{g_R}(\psi))[\xi], \top \rangle \in \mathit{gnd}(R)$, which is generated by a unique ground substitution $\theta_{g_R} \in \mathit{Sol}(C, \mathit{vars}(\phi) \cup \mathit{vars}(\psi))$, the function $f_{g_R} : \mathcal{X}(R) \to [0, 1]$ denotes the *feature function of* g_R, which is defined as:

$$f_{g_R}(\boldsymbol{x}) := \begin{cases} 1 & \text{iff} \quad \boldsymbol{x} \models (\theta_{g_R}(\phi) \wedge \theta_{g_R}(\psi)), \\ 0 & \text{iff} \quad \boldsymbol{x} \models (\neg\theta_{g_R}(\phi) \wedge \theta_{g_R}(\psi)), \\ \xi & \text{iff} \quad \boldsymbol{x} \models (\neg\theta_{g_R}(\psi)). \end{cases} \quad (8)$$

A pair (f, ξ), consisting of a feature function f and its associated expected values ξ, is called a *probabilistic constraint*.

All these definitions are canonically extended to sets $\mathcal{R} = \{R_1, \ldots, R_m\}$ of FO-PCL conditionals, and the set of all FO-PCL models for $\mathcal{R}$ is denoted by $\mathrm{Mod}_{\text{FO-PCL}}(\mathcal{R})$.

Similar to propositional probabilistic conditional logic, the set $\mathrm{Mod}_{\text{FO-PCL}}(\mathcal{R})$ of all FO-PCL models for $\mathcal{R}$ is generally infinite, as it is a convex set: one can show that any convex combination $s_{\boldsymbol{X}(\mathcal{R})}(\boldsymbol{x}) := \delta p_{\boldsymbol{X}(\mathcal{R})}(\boldsymbol{x}) + (1 - \delta) q_{\boldsymbol{X}(\mathcal{R})}(\boldsymbol{x})$ of two models $p_{\boldsymbol{X}(\mathcal{R})}, q_{\boldsymbol{X}(\mathcal{R})} \in \mathrm{Mod}_{\text{FO-PCL}}(\mathcal{R})$, with $\delta \in [0, 1]$, is again an FO-PCL model of $\mathcal{R}$. Therefore, in order to obtain point probabilities for queries, one must select a single model from $\mathrm{Mod}_{\text{FO-PCL}}(\mathcal{R})$. The *principle of maximum entropy* is a suitable model selection criterion, which yields the most unbiased model in $\mathrm{Mod}_{\text{FO-PCL}}(\mathcal{R})$, see [15,16] for axiomatic derivations and a more thorough discussion. Using Lagrange optimization techniques [17], one can show that the FO-PCL model with maximum entropy,

$$p^*_{\boldsymbol{X}(\mathcal{R})} := \operatorname*{argmax}_{p_{\boldsymbol{X}(\mathcal{R})} \in \mathrm{Mod}_{\text{FO-PCL}}(\mathcal{R})} \left(- \sum_{\boldsymbol{x} \in \mathcal{X}(\mathcal{R})} p_{\boldsymbol{X}(\mathcal{R})}(\boldsymbol{x}) \log p_{\boldsymbol{X}(\mathcal{R})}(\boldsymbol{x}) \right), \quad (9)$$

can be represented as

$$p^*_{\boldsymbol{X}(\mathcal{R})}(\boldsymbol{x}) = \frac{1}{Z} \exp \left(\sum_{k=1}^{m} \sum_{g_{R_k} \in \mathit{gnd}(R_k)} \lambda^*_{g_{R_k}} f_{g_{R_k}}(\boldsymbol{x}) \right), \quad (10)$$

hence $p^*_{\boldsymbol{X}(\mathcal{R})}$ is a so-called *Gibbs distribution*. The $f_{g_{R_k}}$ are the feature functions defined for each ground instance $g_{R_k} \in \mathit{gnd}(R_k)$, $R_k \in \mathcal{R}$, according to (8), and $\lambda_{g_{R_k}}$ are the parameters which have to be optimized to find the Gibbs distribution representing the joint probability function with maximum entropy.

In order to compute the entropy-optimal values $\lambda^*_{g_{R_k}}$ of the parameters $\lambda_{g_{R_k}}$, each must be iteratively adjusted until convergence, as there is no closed formula relating the expected values of the feature functions $f_{g_{R_k}}$ to their entropy-optimal parameter values. This is completely different from other formalisms combining first-order logic and probabilistic models, which reuse the same parametrization for each ground instance of a knowledge representation template.

The requirement to iteratively compute the entropy-optimal parameter values might seem to prevent the practical application of FO-PCL from the outset, as there may be hundreds or thousands of ground instances of every FO-PCL conditional R_k, and solving this entropy-optimization problem is computationally infeasible for all but the smallest FO-PCL knowledge bases. But the following examples show that there seem to exist conditions which ensure that some or all ground instances of the same FO-PCL conditional share the same entropy-optimal parameter value. This is advantageous for two reasons: on the one hand, parameter sharing represents the fact that an FO-PCL knowledge base contains identical knowledge pertaining to ground instances sharing the same entropy-optimal parameter value. On the other hand, this allows for computational savings when computing the entropy-optimal parameter values.

4 Examples

This section illustrates the concept of "sharing" the same entropy-optimal parameter value on two examples.

Example 2 ("Common Cold" semantics).

Varying the number of constant symbols in the domain $\mathcal{D}_{\mathrm{CC}} = D^{(Person)}$ of the "Common Cold" example between two and four, different complete ground instances of the first-order probabilistic conditionals CC1–CC3 are created. The expert system shell SPIRIT [18] is then used to compute the entropy-optimal parameter values for these ground instances.

Although the algorithm used for computing the entropy-optimal parameter values has no information about the way the probabilistic conditionals have been generated, i.e. does not know which propositional probabilistic conditionals are ground instances of the same first-order probabilistic conditional, for any fixed set $\mathcal{D}_{\mathrm{CC}} = D^{(Person)}$ of constant symbols, all ground instances of the same first-order probabilistic conditional share the same entropy-optimal parameter value. Due to space restrictions, no table depicting all ground instances and their entropy-optimal parameter value can be shown here. Instead, Table 1 depicts the entropy-optimal value shared by all ground instances of the same FO-PCL conditional, differentiated by the number $\left|D^{(Person)}\right|$ of constant symbols.

The left column of Table 1 shows the number of constants for which the complete ground instance was generated, and the right column presents the entropy-optimal parameter values shared by *all* ground instances of CC1–CC3, respectively.

The "Common Cold" example clearly demonstrates parameter sharing between all ground instances of the same first-order probabilistic conditional. Fur-

Table 1. Entropy-optimal parameter values for the ground instances of the first-order probabilistic conditionals of the "Common Cold" example.

$\left\lvert D^{(Person)}\right\rvert$	Entropy-optimal parameter value
2	$\lambda_{\mathrm{CC1}} = -35.78381574$ $\lambda_{\mathrm{CC2}} = 31.00000000$ $\lambda_{\mathrm{CC3}} = 5.21711313$
3	$\lambda_{\mathrm{CC1}} = -35.26652281$ $\lambda_{\mathrm{CC2}} = 31.00000000$ $\lambda_{\mathrm{CC3}} = 4.47891786$
4	$\lambda_{\mathrm{CC1}} = -34.85353605$ $\lambda_{\mathrm{CC2}} = 31.00000000$ $\lambda_{\mathrm{CC3}} = 3.73955124$

thermore, as there is no reason to assume that there is anything particular or exceptional about the actual ground instances used for showing parameter sharing, we assert that *any* ground instance of the "Common Cold" example has this property of *parametric uniformity*.

However, the following example shows that parameter sharing is no *intrinsic* property of FO-PCL, i.e. it does not necessarily apply to all ground instances of any FO-PCL conditional. This should be obvious, because parameter sharing reflects that the FO-PCL knowledge base contains the same information about the ground instances sharing the same entropy-optimal parameter value. Any conditional representing exceptional knowledge will therefore affect parameter sharing.

Example 3 ("Misanthrope" example and semantics).

The "Misanthrope" example is a simple model about the friendship relations between a (finite) group of people, with one exceptional member, a misanthrope.

In general, people like each other, i.e. if a person V likes another person U, then it is very likely that U likes V, too. But there is one person, identified with the constant symbol a, which does not like other people, i.e. likes them only with a diminutive probability. This can be expressed as follows:

$$\text{MI1: } \langle (\mathit{likes}(U,V) \mid \mathit{likes}(V,U))[0.9], U \neq V \rangle$$
$$\text{MI2: } \langle (\mathit{likes}(a,V))[0.05], V \neq a \rangle$$

Recalling the results of the previous examples, one might assume that all ground instances of MI1 resp. MI2 also share the same entropy-optimal parameter value. However, when generating ground instances for a specific set $D^{(Person)}$ of constant symbols, one can easily see that the "Misanthrope" example is different from the "Common Cold" example.

Table 2 depicts the ground instances of MI1, $g_{\mathrm{MI1},1}$–$g_{\mathrm{MI1},6}$, and MI2, $g_{\mathrm{MI2},1}$ and $g_{\mathrm{MI2},2}$, generated for the constant symbols $D^{(Person)} = \{a, b, c\}$. Whereas all ground instances of MI2 share the same entropy-optimal parameter value,

Table 2. Ground instances and entropy-optimal parameter values of the first-order probabilistic conditionals of the "Misanthrope" example, generated for the constant symbols $D^{(Person)} = \{a, b, c\}$.

Ground instance		Entropy-optimal parameter value
$g_{\mathrm{MI1},1}$	$\langle (likes(a,b) \mid likes(b,a))[0.9], \top \rangle$	8.40249157
$g_{\mathrm{MI1},2}$	$\langle (likes(a,c) \mid likes(c,a))[0.9], \top \rangle$	8.40249157
$g_{\mathrm{MI1},3}$	$\langle (likes(b,a) \mid likes(a,b))[0.9], \top \rangle$	2.32967585
$g_{\mathrm{MI1},4}$	$\langle (likes(b,c) \mid likes(c,b))[0.9], \top \rangle$	2.88175000
$g_{\mathrm{MI1},5}$	$\langle (likes(c,a) \mid likes(a,c))[0.9], \top \rangle$	2.32967585
$g_{\mathrm{MI1},6}$	$\langle (likes(c,b) \mid likes(b,c))[0.9], \top \rangle$	2.88175000
$g_{\mathrm{MI2},1}$	$\langle (likes(a,b)[0.05], \top \rangle$	−5.46553416
$g_{\mathrm{MI2},2}$	$\langle (likes(a,c)[0.05], \top \rangle$	−5.46553416

there are three different entropy-optimal parameter values for different ground instances of MI1.

Example 3 shows that, although not all ground instances of an FO-PCL conditional share the same entropy-optimal parameter value, they can be partitioned into several groups, with the ground instances in each group again sharing the same entropy-optimal parameter value. This leads to the question which properties of an FO-PCL knowledge base give rise to parameter sharing. Ideally, this would be syntactic properties, as these could be verified without actually computing the entropy-optimal parameter values.

Some motivation for the quest for syntactic properties may be derived from Example 3, by observing the way the ground instances of MI1 share ground atoms with ground instances of MI2. Ground instances $g_{\mathrm{MI1},1}$ and $g_{\mathrm{MI1},2}$ both share the ground atom in their conclusion (ground atom $likes(a,b)$ resp. $likes(a,c)$) with one ground instance of MI2. The ground instances $g_{\mathrm{MI1},3}$ and $g_{\mathrm{MI1},5}$ on the other hand share the ground atom in their premise (again, $likes(a,b)$ resp. $likes(a,c)$) with one ground instance of MI2, whereas $g_{\mathrm{MI1},4}$ and $g_{\mathrm{MI1},6}$ do not share any ground atom with a ground instance of MI2. These different patterns of sharing of ground atoms directly correspond to the sharing of entropy-optimal parameter values.

5 Representing Maximum-Entropy Models for FO-PCL

Before we can try to develop syntactic conditions which ensure the semantic notion of parameter sharing, we must first formally define this property.

Given a signature $\Sigma = (S, \mathcal{D}, Pred)$ and a set $\mathcal{R} = \{R_1, \ldots, R_m\}$ of FO-PCL conditionals defined over Σ, two ground instances $g_{R_k}, g'_{R_k} \in gnd(R_k)$ with associated feature functions $f_{g_{R_k}}, f_{g'_{R_k}}$ are called *parametrically equivalent* with respect to the complete ground instance $gnd(\mathcal{R})$ of $\mathcal{R}$ and the maximum-

entropy model $p^*_{\boldsymbol{X}(\mathcal{R})} \in \mathrm{Mod}_{\text{FO-PCL}}(\mathcal{R})$, cf. Equation (10), iff they share the same entropy-optimal parameter values, i.e. $\lambda^*_{g_{R_k}} = \lambda^*_{g'_{R_k}}$.

It is obvious that parametric equivalence is an equivalence relation, whose equivalence classes consist of those ground instances sharing the same entropy-optimal parameter value. This partitioning corresponds to the fact that the FO-PCL knowledge base $\mathcal{R}$ contains the same information about each ground instance in a given equivalence class. Furthermore, the partitioning can be utilized to obtain a simpler entropy-optimization problem: as the same optimization parameter is shared by all ground instances in an equivalence class, only this single parameter has to be computed for all ground instances in the equivalence class.

The partitioning of the ground instances of $R_k = \langle (\phi_k \mid \psi_k)[\xi_k], C_k \rangle \in \mathcal{R}$ can be represented intentionally, by restating C_k as a disjunction $C_{k,1} \vee \ldots \vee C_{k,n_k}$, with $C_{k,i} \wedge C_{k,j} \equiv \bot$, for all $1 \leq i, j \leq n_k$, $i \neq j$. Each constraint formula $C_{k,i}$ represents the ground instances belonging to the same equivalence class of $gnd(R_k)$. To simplify all further expositions, we assume that the single conditional $R_k = \langle (\phi_k \mid \psi_k)[\xi_k], C_k \rangle$ is replaced by n_k conditionals $R_{k,1} = \langle (\phi_k \mid \psi_k)[\xi_k], C_{k,1} \rangle, \ldots, R_{k,n_k} = \langle (\phi_k \mid \psi_k)[\xi_k], C_{k,n_k} \rangle$, each having identical components, except for their constraint formulas. Hence every FO-PCL conditional only has a single associated parameter, instead of one for each of its equivalence classes. Its joint probability function with maximum entropy can then be represented as

$$p^*_{\boldsymbol{X}(\mathcal{R})}(\boldsymbol{x}) = \frac{1}{Z} \exp\left(\sum_{k=1}^{m} \lambda^*_{R_k} \sum_{g_{R_k} \in gnd(R_k)} f_{g_{R_k}}(\boldsymbol{x}) \right).$$

The main idea underlying the syntactic condition for parameter sharing is that, since parametric equivalence represents the fact that an FO-PCL knowledge base $\mathcal{R}$ essentially contains identical knowledge about two ground instances $g_{R_k}, g'_{R_k} \in gnd(R_k)$, $R_k \in \mathcal{R}$, one should be able to "exchange" g_{R_k} and g'_{R_k}, and still obtain the same joint probability function $p^*_{\boldsymbol{X}(\mathcal{R})}$ with maximum entropy. Because $p^*_{\boldsymbol{X}(\mathcal{R})}$ is a Gibbs distribution, it is defined by a unique set $\boldsymbol{\lambda}^*$ of Lagrange multipliers, given some mild conditions[2] on $\mathcal{R}$. Now, if g_{R_k} and g'_{R_k} can be exchanged without changing $p^*_{\boldsymbol{X}(\mathcal{R})}$, this shows that the ground instances g_{R_k} and g'_{R_k} are parametrically equivalent. If arbitrarily chosen $g_{R_k}, g'_{R_k} \in gnd(R_k)$, can be transposed, this shows that R_k is parametrically uniform, i.e. all ground instances of the same FO-PCL conditional R_k share the same entropy-optimal parameter value.

To show parametric equivalence of two ground instances of the same FO-PCL conditional, we use so-called *probabilistic constraint involutions*. Please

[2] It is required that the ground instances resp. their feature functions are linearly independent. However, although the results generally do not hold for linearly dependent ground instances—as the set of Lagrange multipliers is not unique—there seem to exist sets of Lagrange multipliers such that the results also apply to linearly dependent ground instances.

recall that every ground instance $g_{R_k} = \langle (\theta_{g_{R_k}}(\phi_k) \,|\, \theta_{g_{R_k}}(\psi_k))[\xi_k], \top \rangle \in \mathit{gnd}(R_k)$ of a first-order probabilistic conditional $R_k = \langle (\phi_k \,|\, \psi_k)[\xi_k], C_k \rangle \in \mathcal{R}$ is generated by a unique ground substitution $\theta_{g_{R_k}} \in \mathit{Sol}(C_k, \mathit{vars}(\phi_k) \cup \mathit{vars}(\psi_k))$, and has an associated feature function $f_{g_{R_k}}$. Furthermore, the pair $(f_{g_{R_k}}, \xi_k)$ constitutes a probabilistic constraint, and the set of all probabilistic constraints associated with the set $\mathit{gnd}(\mathcal{R})$ of all ground instances is denoted by

$$\mathcal{F}(\mathcal{R}) := \left\{ \left(f_{g_{R_k}}, \xi_k \right) \;\middle|\; R_k = \langle (\phi_k \,|\, \psi_k)[\xi_k], C_k \rangle \in \mathcal{R}, g_{R_k} \in \mathit{gnd}(R_k) \right\}. \tag{11}$$

A probabilistic constraint involution transposes pairs of ground instances of the same FO-PCL conditional $R_k \in \mathcal{R}$. These transpositions can be modeled by a pair $(\pi_{\mathcal{F}(\mathcal{R})}, \pi_{\boldsymbol{X}(\mathcal{R})})$ of permutations, with $\pi_{\mathcal{F}(\mathcal{R})} : \mathcal{F}(\mathcal{R}) \to \mathcal{F}(\mathcal{R})$ and $\pi_{\boldsymbol{X}(\mathcal{R})} : \boldsymbol{X}(\mathcal{R}) \to \boldsymbol{X}(\mathcal{R})$. The permutations $\pi_{\mathcal{F}(\mathcal{R})}$ and $\pi_{\boldsymbol{X}(\mathcal{R})}$ are not independent of each other but are interrelated, as both model the same transpositions of ground instances, only on different levels: $\pi_{\mathcal{F}(\mathcal{R})}$ acts on the set $\mathcal{F}(\mathcal{R})$ of probabilistic constraints, whereas $\pi_{\boldsymbol{X}(\mathcal{R})}$ acts on the set $\boldsymbol{X}(\mathcal{R})$ of random variables. Furthermore, $\pi_{\mathcal{F}(\mathcal{R})}$ and $\pi_{\boldsymbol{X}(\mathcal{R})}$ are *involutions*, i.e. permutations which are their own inverse. Because $\pi_{\mathcal{F}(\mathcal{R})}$ is assumed to only transpose probabilistic constraints corresponding to ground instances of the same FO-PCL conditional, it partitions $\mathcal{F}(\mathcal{R})$ into three sets $\mathcal{F}^{(1)}(\mathcal{R})$, $\mathcal{F}^{(2)}(\mathcal{R})$ and $\mathcal{F}^{(3)}(\mathcal{R})$, such that the following holds:

$$\begin{aligned}
&\forall \left(f^{(1)}_{g_{R_k}}, \xi_k \right) \in \mathcal{F}^{(1)}(\mathcal{R}) \quad \exists \left(f^{(2)}_{g_{R_k}}, \xi_k \right) \in \mathcal{F}^{(2)}(\mathcal{R}) : \\
&\qquad\qquad\qquad\qquad \pi_{\mathcal{F}(\mathcal{R})} \left(\left(f^{(1)}_{g_{R_k}}, \xi_k \right) \right) = \left(f^{(2)}_{g_{R_k}}, \xi_k \right), \\
&\forall \left(f^{(3)}_{g_{R_k}}, \xi_k \right) \in \mathcal{F}^{(3)}(\mathcal{R}) : \qquad \pi_{\mathcal{F}(\mathcal{R})} \left(\left(f^{(3)}_{g_{R_k}}, \xi_k \right) \right) = \left(f^{(3)}_{g_{R_k}}, \xi_k \right).
\end{aligned} \tag{12}$$

The interrelations of $\pi_{\mathcal{F}(\mathcal{R})}$ and $\pi_{\boldsymbol{X}(\mathcal{R})}$ can be illustrated with the help of another involution $\pi_{\mathcal{X}(\mathcal{R})} : \mathcal{X}(\mathcal{R}) \to \mathcal{X}(\mathcal{R})$, which is induced by $\pi_{\boldsymbol{X}(\mathcal{R})}$: whereas $\pi_{\boldsymbol{X}(\mathcal{R})}$ transposes random variables, $\pi_{\mathcal{X}(\mathcal{R})}$ transposes configurations in $\mathcal{X}(\mathcal{R})$, by exchanging the values of those random variables which are transposed by $\pi_{\boldsymbol{X}(\mathcal{R})}$. That is, $\pi_{\mathcal{X}(\mathcal{R})}$ is defined as

$$\pi_{\mathcal{X}(\mathcal{R})}(\boldsymbol{x}) = \pi_{\mathcal{X}(\mathcal{R})}((x_1, \ldots, x_{|\boldsymbol{X}(\mathcal{R})|})) := (x_{\pi_{\boldsymbol{X}(\mathcal{R})}(X_1)}, \ldots, x_{\pi_{\boldsymbol{X}(\mathcal{R})}(X_{|\boldsymbol{X}(\mathcal{R})|})}). \tag{13}$$

Because the value of any feature function $f_{g_{R_k}}$ belonging to a probabilistic constraint $\left(f_{g_{R_k}}, \xi_k \right) \in \mathcal{F}(\mathcal{R})$ only depends on the values of the random variables corresponding to the ground atoms $f_{g_{R_k}}$ is defined over, the interrelations of $\pi_{\mathcal{F}(\mathcal{R})}$ and $\pi_{\boldsymbol{X}(\mathcal{R})}$ resp. $\pi_{\mathcal{X}(\mathcal{R})}$ can be formalized as follows:

$$\begin{aligned}
&\forall \boldsymbol{x} \in \mathcal{X}(\mathcal{R}) \quad \forall \left(f^{(1)}_{g_{R_k}}, \xi_k \right) \in \mathcal{F}^{(1)}(\mathcal{R}) \quad \forall \left(f^{(2)}_{g_{R_k}}, \xi_k \right) \in \mathcal{F}^{(2)}(\mathcal{R}) : \\
&\quad \pi_{\mathcal{F}(\mathcal{R})} \left(\left(f^{(1)}_{g_{R_k}}, \xi_k \right) \right) = \left(f^{(2)}_{g_{R_k}}, \xi_k \right) \quad \Rightarrow \quad f^{(1)}_{g_{R_k}}(\boldsymbol{x}) = f^{(2)}_{g_{R_k}}(\pi_{\mathcal{X}(\mathcal{R})}(\boldsymbol{x})), \\
&\forall \boldsymbol{x} \in \mathcal{X}(\mathcal{R}) \quad \forall \left(f^{(3)}_{g_{R_k}}, \xi_k \right) \in \mathcal{F}^{(3)}(\mathcal{R}) : \qquad f^{(3)}_{g_{R_k}}(\boldsymbol{x}) = f^{(3)}_{g_{R_k}}(\pi_{\mathcal{X}(\mathcal{R})}(\boldsymbol{x})).
\end{aligned} \tag{14}$$

This property states that, for any probabilistic constraints $\left(f^{(1)}_{g_{R_k}}, \xi_k\right) \in \mathcal{F}^{(1)}(\mathcal{R})$ and $\left(f^{(2)}_{g_{R_k}}, \xi_k\right) \in \mathcal{F}^{(2)}(\mathcal{R})$ with $\pi_{\mathcal{F}(\mathcal{R})}\left(\left(f^{(1)}_{g_{R_k}}, \xi_k\right)\right) = \left(f^{(2)}_{g_{R_k}}, \xi_k\right)$, $f^{(1)}_{g_{R_k}}$ evaluates on $\boldsymbol{x} \in \mathcal{X}(\mathcal{R})$ to the same value as $f^{(2)}_{g_{R_k}}$ does on $\pi_{\mathcal{X}(\mathcal{R})}(\boldsymbol{x})$, and vice versa, whereas the probabilistic constraints in $\mathcal{F}^{(3)}(\mathcal{R})$ are not influenced by $\pi_{\mathcal{X}(\mathcal{R})}$.

We illustrate the concept of probabilistic constraints involutions on the "Misanthrope" knowledge base.

Example 4 ("Misanthrope" probabilistic constraint involution).

We can define a probabilistic constraint involution $(\pi_{\mathcal{F}(\mathcal{R}_{\mathrm{MI}})}, \pi_{\boldsymbol{X}(\mathcal{R}_{\mathrm{MI}})})$ for the "Misanthrope" example as

$$\pi_{\mathcal{F}(\mathcal{R}_{\mathrm{MI}})} := \begin{matrix}(g_{\mathrm{MI1},1} & g_{\mathrm{MI1},2})(g_{\mathrm{MI1},3} & g_{\mathrm{MI1},5}) \\ (g_{\mathrm{MI2},1} & g_{\mathrm{MI2},2}),\end{matrix}$$

and

$$\pi_{\boldsymbol{X}(\mathcal{R}_{\mathrm{MI}})} := \begin{matrix}(\mathit{likes}(a,b) & \mathit{likes}(a,c)) \\ (\mathit{likes}(b,a) & \mathit{likes}(c,a)).\end{matrix}$$

Although the "Misanthrope" example is not parametrically uniform, by comparing the feature functions resp. ground instances transposed by $\pi_{\mathcal{F}(\mathcal{R}_{\mathrm{MI}})}$ and their entropy-optimal parameter values depicted in Table 2, one can observe an interesting property: those ground instances which are transposed by $\pi_{\mathcal{F}(\mathcal{R}_{\mathrm{MI}})}$ share the same entropy-optimal parameter value.

This is no coincidence: one can formally proof that those ground instances transposed by a probabilistic constraint involution share the same entropy-optimal parameter value. Though the proof cannot be presented here due to space restrictions, we can further substantiate this proposition by showing that there cannot be a probabilistic constraint involution transposing two ground instances having *different* entropy-optimal parameter values, e.g. $g_{\mathrm{MI1},1}$ and $g_{\mathrm{MI1},3}$. One may try to construct an involution transposing $g_{\mathrm{MI1},1}$ and $g_{\mathrm{MI1},3}$ by defining $\pi_{\mathcal{F}(\mathcal{R}_{\mathrm{MI}})} := (g_{\mathrm{MI1},1} \quad g_{\mathrm{MI1},3})$ and $\pi_{\boldsymbol{X}(\mathcal{R}_{\mathrm{MI}})} := (\mathit{likes}(a,b) \quad \mathit{likes}(b,a))$. Although $g_{\mathrm{MI1},1}$ and $g_{\mathrm{MI1},3}$ are correctly transposed, involution $\pi_{\boldsymbol{X}(\mathcal{R}_{\mathrm{MI}})}$ also affects ground instance $g_{\mathrm{MI2},1} = \langle(\mathit{likes}(a,b))[0.05], \top\rangle$, for which there is no corresponding ground instance of MI2, which would be $\langle(\mathit{likes}(b,a))[0.05], \top\rangle$. Hence there exists no involution $\pi_{\mathcal{F}(\mathcal{R}_{\mathrm{MI}})}$ which transposes $g_{\mathrm{MI1},1}$ and $g_{\mathrm{MI1},3}$, which correlates to the fact that these ground instances have different entropy-optimal parameter values. Similar arguments apply when trying to transpose other ground instances of MI1 with different entropy-optimal parameter values.

For the "Misanthrope" example, a single probabilistic constraint involution is sufficient to proof the parametric equivalence between the ground instances depicted in Table 2. But for other FO-PCL knowledge bases, or larger ground instances of the "Misanthrope" knowledge base, more probabilistic constraint involutions are necessary in order to show parametric equivalence between all

ground instances of an equivalence class of an FO-PCL conditional. If all ground instances of an equivalence class shall share the same entropy-optimal parameter value, they should all be interchangeable, i.e. one should be able to arbitrarily permute the members of an equivalence class, any should still obtain the same maximum-entropy model. This can be done with a set of probabilistic constraint involutions, which can be composed to yield any permutation for the elements of any equivalence class. As this is a rather straightforward extension, we don not go into details, but demonstrate how the "Misanthrope" knowledge base can be transformed in order to obtain a parametrically uniform knowledge base.

As stated in the discussion at the end of Section 4, it is the different "patterns" according to which ground instances of MI1 share ground atoms—i.e. random variables—with ground instances of MI2, which cause the parametric non-uniformity of this FO-PCL knowledge base. For example, the parametrically equivalent ground instances $g_{\mathrm{MI1},1}$ and $g_{\mathrm{MI1},2}$ (see Table 2) share the ground atom in their conclusion with one ground instance of MI2. The parametrically equivalent ground instances $g_{\mathrm{MI1},3}$ and $g_{\mathrm{MI1},5}$ share the ground atom in their premise with one ground instance of MI2, and parametrically equivalent ground instances $g_{\mathrm{MI1},4}$ and $g_{\mathrm{MI1},6}$ share none of their ground atoms with a ground instance of MI2.

It is these different "patterns" of sharing ground atoms with ground instances of MI2 which have to be taken into account when transforming MI1 in order to obtain a parametrically uniform knowledge base. This transformation is accomplished by replacing conditional MI1 by three conditionals MI1-1–MI1-3, which yields the following FO-PCL knowledge base:

$$\begin{aligned}
&\text{MI1-1: } \langle (\mathit{likes}(U,V) \mid \mathit{likes}(V,U))[0.9], U \neq V \wedge U \neq a \wedge V \neq a \rangle \\
&\text{MI1-2: } \langle (\mathit{likes}(a,V) \mid \mathit{likes}(V,a))[0.9], V \neq a \rangle \\
&\text{MI1-3: } \langle (\mathit{likes}(U,a) \mid \mathit{likes}(a,U))[0.9], U \neq a \rangle \\
&\text{MI2 } : \langle (\mathit{likes}(a,V))[0.05], V \neq a \rangle
\end{aligned}$$

It can be easily seen that the ground instances of MI1-1–MI1-3 form a partitioning of the ground instances of the original FO-PCL conditional MI1, but each of the conditionals MI1-1–MI1-3 is parametrically uniform.

We assert that similar transformations can be applied on *any* FO-PCL knowledge base in order to make it parametrically uniform, which would greatly simplify the entropy-optimization problem for FO-PCL.

6 Conclusions

This paper has introduced FO-PCL, a first-order extension of probabilistic conditional logic, which allows for the adequate representation of relational probabilistic *if-then*-rules. Similar to most other formalisms combining a subset of first-order logic and probabilistic graphical models, FO-PCL knowledge bases are propositionalized, which in case of first-order probabilistic conditional logic yields a complex optimization problem. We have developed syntactic conditions which can be utilized to simplify this optimization problem, by showing which

ground instances of an FO-PCL conditional share the same entropy-optimal parameter value.

The results presented are only a first step towards an applicable FO-PCL. Further work is necessary to develop an algorithm for transforming arbitrary FO-PCL knowledge bases into parametrically uniform sets of conditionals. In addition to this, efficient algorithms for solving the entropy-optimization problem have to be devised.

Acknowledgments. The research reported here was partly supported by the DFG – Deutsche Forschungsgemeinschaft (grant BE 1700/7-1).

References

1. Cheeseman, P.: In defense of probability. In Joshi, A.K., ed.: Proceedings of the Nineth International Joint Conference on Artificial Intelligence, IJCAI 85, Morgan Kaufmann (1985) 1002–1009
2. Lindley, D.V.: The probability approach to the treatment of uncertainty in artificial intelligence and expert systems. Statistical Science **2**(1) (1987) 3–44
3. Pearl, J.: Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann Publishers (1988)
4. Cowell, R.G., Dawid, A.P., Lauritzen, S.L., Spiegelhalter, D.J.: Probabilistic Networks and Expert Systems. Springer (1999)
5. Getoor, L., Taskar, B., eds.: Introduction to Statistical Relational Learning. MIT Press (2007)
6. de Salvo Braz, R., Amir, E., Roth, D.: A survey of first-order probabilistic models. In Holmes, D.E., Jain, L.C., eds.: Innovations in Bayesian Networks. Volume 156 of Studies in Computational Intelligence. Springer (2008) 289–317
7. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning **62**(1–2) (2006) 107–136
8. Calabrese, P.G.: Deduction and inference using conditional logic and probability. In Goodman, I.R., Gupta, M.M., Nguyen, H.T., Rogers, G.S., eds.: Conditional Logic in Expert Systems. North-Holland (1991) 71–100
9. Nguyen, H.T., Goodman, I.R.: On modeling of if-then rules for probabilistic inference. International Journal of Intelligent Systems **9** (1994) 411–418
10. Rödder, W., Kern-Isberner, G.: Representation and extraction of information by probabilistic logic. Information Systems **21**(8) (1997) 637–652
11. Kern-Isberner, G., Lukasiewicz, T.: Combining probabilistic logic programming with the power of maximum entropy. Artificial Intelligence **157** (2004) 139–202
12. Comon, H., Lescanne, P.: Equational problems and disunification. Journal of Symbolic Computation **7**(3/4) (1989) 371–425
13. Buntine, W.L., Bürckert, H.J.: On solving equations and disequations. Journal of the Association for Computing Machinery **41**(1) (1994) 591–629
14. Halpern, J.Y.: An analysis of first-order logics of probability. Artificial Intelligence **46**(3) (1990) 311–350
15. Jaynes, E.T.: Where do we stand on maximum entropy. In Levine, R.D., Tribus, M., eds.: The Maximum Entropy Formalism, MIT Press (1978)
16. Shore, J.E., Johnson, R.W.: Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. IEEE Transactions on Information Theory **26**(1) (1980) 26–37
17. Kapur, J.N., Kesavan, H.K.: Entropy Optimization Principles with Applications. Academic Press, Inc. (1992)
18. Rödder, W., Reucher, E., Kulmann, F.: Features of the expert-system-shell SPIRIT. Logic Journal of IGPL **14**(3) (2006) 483–500

Towards a Toolbox for Relational Probabilistic Knowledge Representation, Reasoning, and Learning

Marc Finthammer[1], Sebastian Loh[2], and Matthias Thimm[2]

[1] Department of Computer Science, FernUniversität in Hagen, Germany
[2] Department of Computer Science, Technische Universität Dortmund, Germany

Abstract. This paper presents KReator, a versatile and easy-to-use toolbox for statistical relational learning currently under development. The research on combining probabilistic models and first-order theory put forth a lot of different approaches in the past few years. While every approach has advantages and disadvantages the variety of prototypical implementations make thorough comparisons of different approaches difficult. KReator aims at providing a common and simple interface for representing, reasoning, and learning with different relational probabilistic approaches. We give an overview on the system architecture of KReator and illustrate its usage.

1 Introduction

Probabilistic inductive logic programming (or *statistical relational learning*) is a very active field in research at the intersection of logic, probability theory, and machine learning, see [1, 2] for some excellent overviews. This area investigates methods for representing probabilistic information in a relational context for both reasoning and learning. Many researchers developed liftings of propositional probabilistic models to the first-order case in order to take advantage of methods and algorithms already developed. Among these are the well-known Bayesian logic programs [3] and Markov logic networks [4] which extend respectively Bayes nets and Markov nets [5] and are based on knowledge-based model construction [6]. Other approaches also employ Bayes nets for their theoretical foundation like logical Bayesian networks [7] and relational Bayesian networks [8]; or they are influenced by other fields of research like probabilistic relational models [9] by database theory and P-log [10] by answer set programming. There are also some few approaches to apply maximum entropy methods to the relational case [11, 12]. But because of this variety of approaches and the absence of a common interface there are only few comparisons of different approaches, see for example [13, 14].

In this paper we describe the KReator toolbox, a versatile integrated development environment for knowledge engineering in the field of statistical relational learning. KReator is currently under development and part of the ongoing KReate project[3] which aims at developing a common methodology for

[3] `http://www.fernuni-hagen.de/wbs/research/kreate/index.html`

learning, modelling and inference in a relational probabilistic framework. As statistical relational learning is a (relatively) young research area there are many different proposals for integrating probability theory in first-order logic, some of them mentioned above. Although many researchers have implementations of their approaches available, most of these implementations are prototypical, and in order to compare different approaches one has to learn the usage of different tools. KReator aims at providing a common interface for different approaches to statistical relational learning and to support the researcher and knowledge engineer in developing knowledge bases and using them in a common and easy-to-use fashion. Currently, the development of KReator is still in a very early stage but already supports Bayesian logic programs, Markov logic networks, and in particular a new approach for using maximum entropy methods in a relational context [11].

The rest of this paper is organized as follows. In Sec. 2 we give an overview on the approaches of statistical relational learning that are currently supported by KReator, i. e. Bayesian logic programs, Markov logic networks, and the relational maximum entropy approach. We go on in Sec. 3 with presenting the system architecture of KReator and motivate the main design choices. In Sec. 4 we give a short manual-style overview on the usage of KReator and in Sec. 5 we give some hints on future work and conclude.

2 Relational Probabilistic Knowledge Representation

In the following we give some brief overview on frameworks for relational probabilistic reasoning that are already implemented in KReator. These are Bayesian logic programs originally due to Kersting et. al. [3], Markov logic networks originally due to Domingos et. al. [4], and a framework employing reasoning with maximum entropy methods that is currently in development [11]. We illustrate the use of these frameworks on a common example, the well-known burglary example [5, 1].

Example 1. We consider a scenario where someone—let's call him *James*—is on the road and gets a call from his neighbor saying that the alarm of James' house is ringing. James has some uncertain beliefs about the relationships between burglaries, types of neighborhoods, natural disasters, and alarms. For example, he knows that if there is a tornado warning for his home place, then the probability of a tornado triggering the alarm of his house is 0.9. A reasonable information to infer from his beliefs and the given information is "What is the probability of an actual burglary?".

2.1 Bayesian Logic Programs

Bayesian logic programming is an approach to combine logic programming and Bayesian networks [3]. Bayesian logic programs (BLPs) use a standard logic programming language and attach to each logical clause a set of probabilities, which define a conditional probability distribution of the head of the clause given specific instantiations of the body of the clause.

In contrast to first-order logic, BLPs employ an extended form of predicates and atoms. In BLPs, *Bayesian predicates* are predicates that feature an arbitrary set as possible states that are not necessarily the boolean values $\{\mathsf{true}, \mathsf{false}\}$. For example, the Bayesian predicate *bloodtype/1* may represent the blood type of a person using the possible states $S(bloodtype) = \{a, b, ab, 0\}$ [3]. Analogously to first-order logic, Bayesian predicates can be instantiated to *Bayesian atoms* using constants and variables and then each ground Bayesian atom represents a single random variable. If A is a Bayesian atom of the Bayesian predicate p we set $S(A) = S(p)$.

Definition 1 (Bayesian Clause, Conditional Probability Distribution). *A* Bayesian clause c *is an expression* $(H \mid B_1, \ldots, B_n)$ *with Bayesian atoms* $H, B_1, \ldots, B_n$*. With a Bayesian clause* c *with the form* $(H \mid B_1, \ldots, B_n)$ *we associate a function* $\mathsf{cpd}_c : S(H) \times S(B_1) \times \ldots \times S(B_n) \to [0, 1]$ *that fulfills*

$$\forall b_1 \in S(B_1), \ldots, b_n \in S(B_n) : \sum_{h \in S(H)} \mathsf{cpd}_c(h, b_1, \ldots, b_n) = 1 \quad .$$

We call cpd_c *a* conditional probability distribution*. Let* CPD_p *denote the set of all conditional probability distributions* $\{\mathsf{cpd}_{H|B_1,\ldots B_n} \mid H$ *is an atom of* $p\}$*.*

A function cpd_c for a Bayesian clause c expresses the conditional probability distribution $P(\mathsf{head}(c) \mid \mathsf{body}(c))$ and thus partially describes an underlying probability distribution P.

Example 2. We represent Ex. 1 as a set $\{c_1, c_2, c_3\}$ of Bayesian clauses with

$$\begin{aligned} c_1 &: \quad (alarm(\mathsf{X}) \mid burglary(\mathsf{X})) \\ c_2 &: \quad (alarm(\mathsf{X}) \mid lives\_in(\mathsf{X}, \mathsf{Y}), tornado(\mathsf{Y})) \\ c_3 &: \quad (burglary(\mathsf{X}) \mid neighborhood(\mathsf{X})) \end{aligned}$$

where $S(tornado/1) = S(lives\_in/2) = S(alarm) = S(burglary) = \{\mathsf{true}, \mathsf{false}\}$ and $S(neighborhood) = \{\mathsf{good}, \mathsf{average}, \mathsf{bad}\}$. For each Bayesian clause c_i, we define a function cpd_{c_i} which expresses our subjective beliefs, e. g., for clause c_2 we define

$$\begin{aligned} \mathsf{cpd}_{c_2}(\mathsf{true}, \mathsf{true}, \mathsf{true}) &= 0.9 & \mathsf{cpd}_{c_2}(\mathsf{true}, \mathsf{true}, \mathsf{false}) &= 0.01 \\ \mathsf{cpd}_{c_2}(\mathsf{true}, \mathsf{false}, \mathsf{true}) &= 0 & \mathsf{cpd}_{c_2}(\mathsf{true}, \mathsf{false}, \mathsf{false}) &= 0 \\ \mathsf{cpd}_{c_2}(\mathsf{false}, \mathsf{true}, \mathsf{true}) &= 0.1 & \mathsf{cpd}_{c_2}(\mathsf{false}, \mathsf{true}, \mathsf{false}) &= 0.99 \\ \mathsf{cpd}_{c_2}(\mathsf{false}, \mathsf{false}, \mathsf{true}) &= 1 & \mathsf{cpd}_{c_2}(\mathsf{false}, \mathsf{false}, \mathsf{false}) &= 1 \end{aligned}$$

Considering clauses c_1 and c_2 in Ex. 2 one can see that it is possible to have multiple clauses with the same head. BLPs facilitate *combining rules* in order to aggregate probabilities that arise from applications of different Bayesian clauses. A combining rule cr_p for a Bayesian predicate p/n is a function $\mathsf{cr}_p : \mathfrak{P}(\mathsf{CPD}_p) \to \mathsf{CPD}_p$ that assigns to the conditional probability distributions of a set of Bayesian clauses a new conditional probability distribution that represents the *joint* probability distribution obtained from aggregating the given

clauses[4]. For example, given clauses $c_1 = (b(X) \mid a_1(X))$ and $c_2 = (b(X) \mid a_2(X))$ the result $f = \mathsf{cr}_b(\{\mathsf{cpd}_{c_1}, \mathsf{cpd}_{c_2}\})$ of the combining rule cr_b is a function $f : S(b) \times S(a_1) \times S(a_2) \to [0, 1]$. Appropriate choices for such functions are *average* or *noisy-or*, cf. [3].

Example 3. We continue Ex. 2. Suppose *noisy-or* to be the combining rule for *alarm*. Then the joint conditional probability distribution $\mathsf{cpd}_{c'}$ for $c' = (alarm(\mathsf{X}) \mid burglary(\mathsf{X}), lives\_in(\mathsf{X}, \mathsf{Y}), tornado(\mathsf{Y}))$ can be computed via

$$\mathsf{cpd}_{c'}(t_1, t_2, t_3, t_4) = Z * (1 - (1 - \mathsf{cpd}_{c_1}(t_1, t_2)) * (1 - \mathsf{cpd}_{c_2}(t_1, t_3, t_4)))$$

for any $t_1, t_2, t_3, t_4 \in \{\mathsf{true}, \mathsf{false}\}$. Here, Z is a normalizing constant for maintaining the property of conditional probability distributions to sum up to one for any specific head value.

Now we are able to define Bayesian logic programs as follows.

Definition 2 (Bayesian Logic Program). *A* Bayesian logic program B *is a tuple* $B = (C, D, R)$ *with a (finite) set of Bayesian clauses* $C = \{c_1, \ldots, c_n\}$*, a set of conditional probability distributions (one for each clause in* C*)* $D = \{\mathsf{cpd}_{c_1}, \ldots, \mathsf{cpd}_{c_n}\}$*, and a set of combining functions (one for each Bayesian predicate appearing in* C*)* $R = \{\mathsf{cr}_{p_1}, \ldots, \mathsf{cr}_{p_m}\}$.

Semantics are given to Bayesian logic programs via transformation into the propositional case, i. e. into Bayesian networks [5]. Given a specific (finite) universe U a Bayesian network BN can be constructed by introducing a node for every grounded Bayesian atom in B. Using the conditional probability distributions of the grounded clauses and the combining rules of B a (joint) conditional probability distribution can be specified for any node in BN. If BN is acyclic this transformation uniquely determines a probability distribution P on the grounded Bayesian atoms of B which can be used to answer queries.

A detailed description of the above (declarative) semantics and an equivalent procedural semantics which is based on SLD resolution are given in [3].

2.2 Markov Logic Networks

Markov logic [4] establishes a framework which combines Markov networks [5] with first-order logic to handle a broad area of statistical relational learning tasks. The Markov logic syntax complies with first-order logic[5], however each formula is quantified by an additional weight value. The semantics of a set of Markov logic formulas is explained by a probability distribution over possible worlds. A possible world assigns a truth value to every possible ground atom (constructible from the set of predicates and the set of constants). The probability distribution is calculated as a log-linear model over weighted ground formulas.

[4] $\mathfrak{P}(S)$ denotes the power set of a set S.

[5] Although Markov logic also covers functions, we will omit this fact, and only consider constants.

The fundamental idea in Markov logic is that first-order formulas are not handled as hard constraints. Instead, each formula is more or less softened depending on its weight. So a possible world *may* violate a formula without necessarily receiving a zero probability. Rather a world is more probable, the less formulas it violates. A formula's weight specifies how strong the formula is, i. e. how much the formula influences the probability of a satisfying world versus a violating world. This way, the weights of all formulas influence the determination of a possible world's probability in a complex manner. One clear advantage of this approach is that Markov logic can directly handle contradictions in a knowledge base, since the (contradictious) formulas are weighted against each other anyway. Furthermore, by assigning appropriately high weight values to certain formulas, it can be enforced that these formulas will be handled as hard constraints, i. e. any world violating such a hard formula will have a zero probability. Thus, Markov logic also allows the processing of purely logical first-order formulas.

Definition 3 (Markov logic network). *A* Markov logic network (MLN) L *is a set of first-order logic formulas* F_i*, where each formula* F_i *is quantified by a real value* w_i*, its* weight*. Together with a set of constants* C *it defines a* Markov network $M_{L,C}$ *as follows:*

- $M_{L,C}$ *contains a node for each possible grounding of each predicate appearing in* L*.*
- $M_{L,C}$ *contains an edge between two nodes (i. e. ground atoms) iff the ground atoms appear together in at least one grounding of one formula in* L*.*
- $M_{L,C}$ *contains one feature (function) for each possible grounding of each formula* F_i *in* L*. The value of the feature for a possible world* x *is 1, if the ground formula is true for* x *(and 0 otherwise). Each feature is weighted by the weight* w_i *of its respecting formula* F_i*.*

According to the above definition, a MLN (i. e. the weighted formulas) defines a template for constructing *ground Markov networks*. For a different set C' of constants, a different ground Markov network $M_{L,C'}$ emerges from L. These ground Markov networks may vary in size, but their general structure is quite similar, e. g. the groundings of a formula F_i have the weight w_i in any ground Markov network of L. The ground Markov network $M_{L,C}$ specifies

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right)$$

as the probability distribution over possible worlds x (whereas Z is a normalization factor). For each formula F_i, the binary result values of its feature functions have been incorporated into the counting function $n_i(x)$, so $n_i(x)$ compactly expresses the number of true groundings of F_i in the possible world x.

Example 4. In the following example, we model the relations described in Ex. 1 as a MLN (using the Alchemy syntax [15] for MLN files). The "!" operator used in the predicate declarations of *lives_in* and *neighborhood* enforces that the respective variables will have mutually exclusive and exhaustive values, i. e. that every person lives in exactly one town and one neighborhood (in

terms of ground atoms). The weights of the formulas are to be understood exemplary (since "realistic" weights cannot be estimated just like that, but requires learning from data). We declare the typed predicates $alarm(person)$, $neighborhood(person, hood\_state!)$, $lives\_in(person, town!)$, $burglary(person)$, the types and constants $person = \{James, Carl\}$, $town = \{Freiburg, Yorkshire, Austin\}$, $hood\_state = \{Bad, Average, Good\}$, and add the following weighted formulas:

$$\begin{aligned} 2.2 \quad & burglary(x) && \Rightarrow alarm(x) \\ 2.2 \quad & lives\_in(x, y) \wedge tornado(y) && \Rightarrow alarm(x) \\ -0.8 \quad & neighborhood(x, Good) && \Rightarrow burglary(x) \\ -0.4 \quad & neighborhood(x, Average) && \Rightarrow burglary(x) \\ 0.4 \quad & neighborhood(x, Bad) && \Rightarrow burglary(x) \end{aligned}$$

2.3 Relational Maximum Entropy

In this paper we also consider a specific approach for reasoning under maximum entropy on first-order probabilistic conditional logic [11]. Knowledge is captured in RME using probabilistic conditionals as in probabilistic conditional logic, cf. [16].

Definition 4 (RME conditional). *A RME conditional $r = (\phi \mid \psi)[\alpha][c]$ consists of a head literal ϕ, a list of n body literals $\psi = \psi_1, \ldots, \psi_n$, a real value $\alpha \in [0, 1]$, and a list of* meta-constraints *$c = c_1, \ldots, c_m$, which allows the restriction of the substitution for certain variables. A meta-constraint is either an expression of the form $X \neq Y$ or $X \notin \{k_1, \ldots, k_l\}$, with variables X, Y and $\{k_1, \ldots, k_l\} \subseteq U$. A conditional r is called* ground *iff r contains no variables. The set of all RME conditionals is determined as the language $(\mathcal{L} \mid \mathcal{L})^{rel}$ and the set of all ground conditionals is referred to by $(\mathcal{L} \mid \mathcal{L})^{rel}_U$.*

Definition 5 (RME knowledge base). *A RME knowledge base KB is a quadruple $KB = (S, U, P, R)$ with a finite set of* sorts *S, a finite set of constants U, a finite set of predicates P, and a finite set of RME conditionals R. Any constant of the* universe *U is associated with one sort in S and U_σ determines the constants with sort σ. Each argument of a predicate $p(\sigma_1, \ldots, \sigma_k) \in P$ is also associated with a sort in S, $\sigma_i \in S, 1 \leq i \leq k$. Furthermore, all variables $Var(R)$ occurring in R are associated with a sort in S, too. Constants and variables are referred to as terms t.*

Example 5. We represent Ex. 1 as a RME knowledge base KB which consists of sorts $S = \{Person, Town, Status\}$, constants $U_{Person} = \{carl, stefan\}$ of sort $Person$, $U_{Town} = \{freiburg, yorkshire, austin\}$ of sort $Town$, $U_{Status} = \{bad, average, good\}$ of sort $Status$, predicates $P = \{alarm(Person)$, $burglary(Person)$, $lives\_in(Person, Town)$, $neighbourhood(Person, Status)\}$, and conditionals $R = \{c_1, \ldots, c_7\}$.

$$
\begin{aligned}
c_1 &= (\mathit{alarm}(\mathsf{X}) \mid \mathit{burglary}(\mathsf{X}))\ [0.9] \\
c_2 &= (\mathit{alarm}(\mathsf{X}) \mid \mathit{lives\_in}(\mathsf{X}, \mathsf{Y}), \mathit{tornado}(\mathsf{Y}))\ [0.9]\ \} \\
c_3 &= (\mathit{burglary}(\mathsf{X}) \mid \mathit{neighborhood}(\mathsf{X}, \mathit{bad}))\ [0.6] \\
c_4 &= (\mathit{burglary}(\mathsf{X}) \mid \mathit{neighborhood}(\mathsf{X}, \mathit{average}))\ [0.4] \\
c_5 &= (\mathit{burglary}(\mathsf{X}) \mid \mathit{neighborhood}(\mathsf{X}, \mathit{good}))\ [0.3] \\
c_6 &= (\mathit{neighborhood}(\mathsf{X}, \mathsf{Z}) \mid \mathit{neighborhood}(\mathsf{X}, \mathsf{Y}))\ [0.0]\ [\mathsf{Y} \neq \mathsf{Z}] \\
c_7 &= (\mathit{lives\_in}(\mathsf{X}, \mathsf{Z}) \mid \mathit{lives\_in}(\mathsf{X}, \mathsf{Y}))\ [0.0]\ [\mathsf{Y} \neq \mathsf{Z}]
\end{aligned}
$$

Notice, that conditionals c_6 and c_7 ensure mutual exclusion of the states for literals of *neighborhood* and *lives_in*.

Semantics are given to RME knowledge bases by grounding R with a *grounding operator* (GOP)[6] to a propositional probabilistic knowledge base and calculating the probability distribution with maximum entropy $P^{ME}_{\mathcal{G}_\chi(R)}$. A GOP (see Fig. 1) is a type of substitution pattern that facilitates the modeling of exceptional knowledge. For example, we could add the exceptional rule

$$c_8 = (\mathit{alarm}(\mathsf{X}) \mid \mathit{lives\_in}(\mathsf{X}, \mathit{freiburg}), \mathit{tornado}(\mathit{freiburg}))\ [0.1]$$

to KB in order to model that in Freiburg tornados are usually not strong enough to cause an alarm. Then the GOP $\mathcal{G}_{priority}$ would prefer all instances of c_8 above all instances of c_2, in which the constant *freiburg* occurs (for details see [11]).

After R is grounded, $\mathcal{G}_\chi(R)$ is treated as a propositional knowledge base and $P^{ME}_{\mathcal{G}_\chi(R)}$ can be calculated as in the propositional case [16]. A RME conditional $Q \in (\mathcal{L} \mid \mathcal{L})^{rel}$ is *fulfilled under the grounding* $\mathcal{G}_\chi(R)$ by the RME knowledge base R if the following hold:

$$R \models^{ME}_{\mathcal{G}_\chi} Q \iff P^{ME}_{\mathcal{G}_\chi(R)} \models \mathcal{G}_\chi(Q) \iff \forall q \in \mathcal{G}_\chi(Q) : P^{ME}_{\mathcal{G}_\chi(R)} \models q.$$

The RME inference process can be divided into three steps: 1.) ground the KB with a certain GOP $\mathcal{G}_\chi$, 2.) calculate the probability distribution $P^{ME}_{\mathcal{G}_\chi(R)}$ with maximum entropy for the grounded instance $\mathcal{G}_\chi(R)$, and 3.) calculate all probabilistic implications of $P^{ME}_{\mathcal{G}_\chi(R)}$.

A more elaborated overview on the framework of relational maximum entropy as introduced here is given in [11].

[6] In [11] the *naive*-, *cautious*-, *conservative*-, and *priority*-grounding strategies are presented and analyzed.

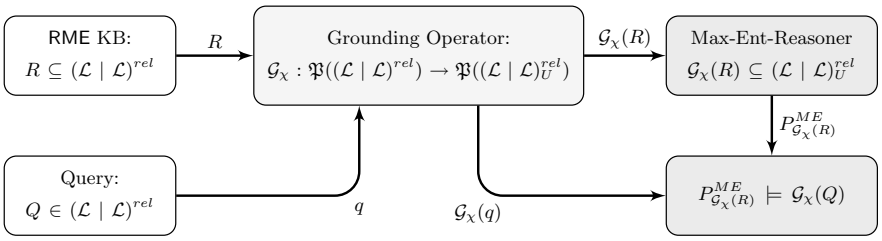

Fig. 1. RME semantics overview. A RME KB R is transformed into a propositional representation by the GOP $\mathcal{G}_\chi$. The result $\mathcal{G}_\chi(R)$ is used to calculate the Max-Ent-distribution $P^{ME}_{\mathcal{G}_\chi(R)}$ in order to answer the ground query $\mathcal{G}_\chi(q)$.

3 System Architecture

KReator[7] is an integrated development environment for representing, reasoning, and learning with relational probabilistic knowledge. Still being in development KReator aims to become a versatile toolbox for researchers and knowledge engineers in the field of statistical relational learning. KReator is written in Java and thus is designed using the object-oriented programming paradigm. It facilitates several architectural and design patterns such as model-view control, abstract factories, and command patterns. Central aspects of the design of KReator are *modularity*, *extensibility*, *usability*, *reproducibility*, and its intended application in scientific research.

Modularity and Extensibility KReator is modular and extensible with respect to several components. In the following we discuss just two important aspects. First, KReator separates between the internal logic and the user interface using an abstract command structure. Each top-level functionality of KReator is internally represented and encapsulated in an abstract KReatorCommand. Consequently, the user interface can be exchanged or modified in an easy and unproblematic way, because it is separated from the internal program structure by this KReatorCommand layer. As a matter of fact, the current version of KReator features both a graphical user interface and a command line interface (the KReator *console*) which processes commands in KReatorScript syntax (see Sec. 4.1). Second, KReator was designed to support many different approaches for relational knowledge representation, cf. Sec. 2. As a consequence, KReator features very abstract notions of concepts like knowledge bases, queries and data sets that can be implemented by a specific approach. At the moment, KReator supports knowledge representation using Bayesian logic programs (BLPs), Markov logic networks (MLNs), and the relational maximum entropy (RME) approach described in Sec. 2.3. Other formalisms will be integrated in the near future.

[7] The "KR" in KReator stands for "Knowledge Representation" and the name KReator indicates its intended usage as a development environment for knowledge engineers.

Usability and Reproducibility An important design aspect of KReator and especially of the graphical user interface is usability. While prototypical implementations of specific approaches to relational probabilistic knowledge representation (and approaches for any problem in general) are essential for validating results and evaluation, these software solutions are often very hard to use and differ significantly in their usage. Especially when one wants to compare different solutions these tools do not offer an easy access for new users. KReator features a common and simple interface to different approaches of relational probabilistic knowledge representation within a single application.

Application in Scientific Research Both usability and reproducibility are important aspects when designing a tool for conducting scientific research. Besides that, other important features are also provided within KReator. For example, KReator can export knowledge base files as formatted LaTeX output, making the seamless processing of example knowledge bases in scientific publications very convenient. KReator records every user operation (no matter whether it was caused by GUI interaction or by a console input) and its result in a *report*. Since all operations are reported in KReatorScript syntax, the report itself represents a valid script. Therefore the whole report or parts of it can be saved as a KReatorScript file which can be executed anytime to repeat the recorded operations.

Used frameworks KReator makes use of well-established software frameworks to process some of the supported knowledge representation formalisms. Performing inference on MLNs is handled entirely by the Alchemy software package [15], a console-based tool for processing Markov logic networks. Alchemy is open source software developed by the inventors of Markov logic networks and can freely be obtained on `http://alchemy.cs.washington.edu/`. To process ground RME knowledge bases, an appropriate reasoner for maximum entropy must be utilized. KReator does not directly interact with a certain reasoner. Instead, KReator uses a so-called ME-adapter to communicate with a (quite arbitrary) MaxEnt-reasoner. Currently, such an adapter is supplied for the SPIRIT reasoner [16]. SPIRIT is a tool for processing (propositional) conditional probabilistic knowledge bases using maximum entropy methods and can be obtained on `http://www.fernuni-hagen.de/BWLOR/spirit_int/`. An appropriate adapter for the MECoRe reasoner [17] has also been developed.

4 Usage

KReator comes with a graphical user interface and an integrated console-based interface. The main view of KReator (see Fig. 2) is divided into the menu and toolbars and four main panels: the project panel, the editor panel, the outline panel, and the console panel.

The project panel KReator structures its data into projects which may contain knowledge bases, scripts written in KReatorScript (see below), query collections for knowledge bases, and sample/evidence files. Although all types of files

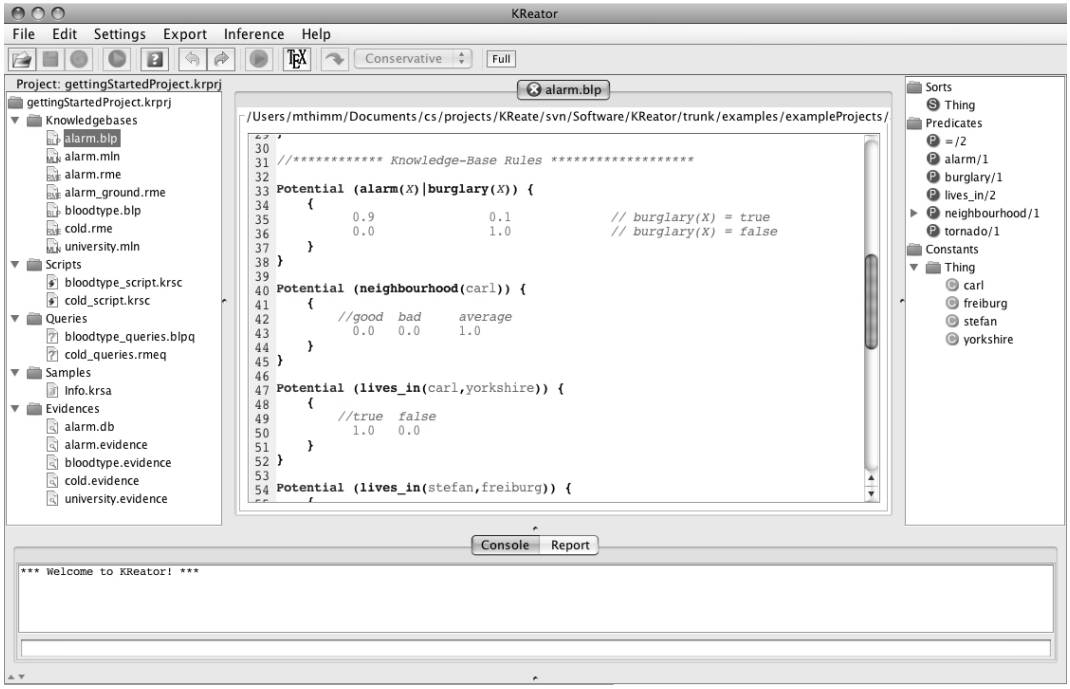

Fig. 2. KReator – Main window

can be opened independently in KReator, projects can help the knowledge engineer to organize his work. The project panel of KReator (seen in the upper left in Fig. 2) gives a complete overview on the project the user is currently working on.

The editor panel All files supported by KReator can be viewed and edited in the editor panel (seen in the upper middle in Fig. 2). Multiple files can be opened at the same time and the editor supports editing knowledge bases and the like with syntax-highlighting, syntax check, and other features normally known from development environments for programming languages.

The outline panel The outline panel (seen in the upper right in Fig. 2) gives an overview on the currently viewed file in the editor panel. If the file is a knowledge base the outline shows information on the logical components of the knowledge base, such as used predicates (and, in case of BLPs, their states), constants, and sorts (if the knowledge base uses a typed language).

The console panel The console panel (seen at the bottom in Fig. 2) contains two tabs, one with the actual console interface and one with the *report*. The console can be used to access nearly every KReator functionality just using textual commands, e. g. querying knowledge bases, open and saving file, and so on. The console is a live interpreter for KReatorScript, the scripting language also used for writing scripts (see below). The console panel also contains the report tab. Every action executed in KReator, e. g. opening a file in the graphical

user interface or querying a knowledge base from the console, is recorded as a KReatorScript command in the report. The whole report or parts of it can easily be saved as script file and executed again when experiments have to be repeated and results have to be reproduced.

4.1 The KReatorScript Language

The KReatorScript language incorporates all those commands which can be processed by the console and by script files as well (see Fig. 3 for some KReatorScript lines). Since there are commands available for all high-level KReator functionalities, every sequence of working steps can be expressed as an appropriate command sequence in a KReatorScript file. Thus, the (re-)utilization of scripts can clearly increase the efficiency and productivity when working with KReator. As mentioned above, the input console and report work hand in hand with KReator's scripting functionality, making the KReatorScript language a strong instrument in the whole working process.

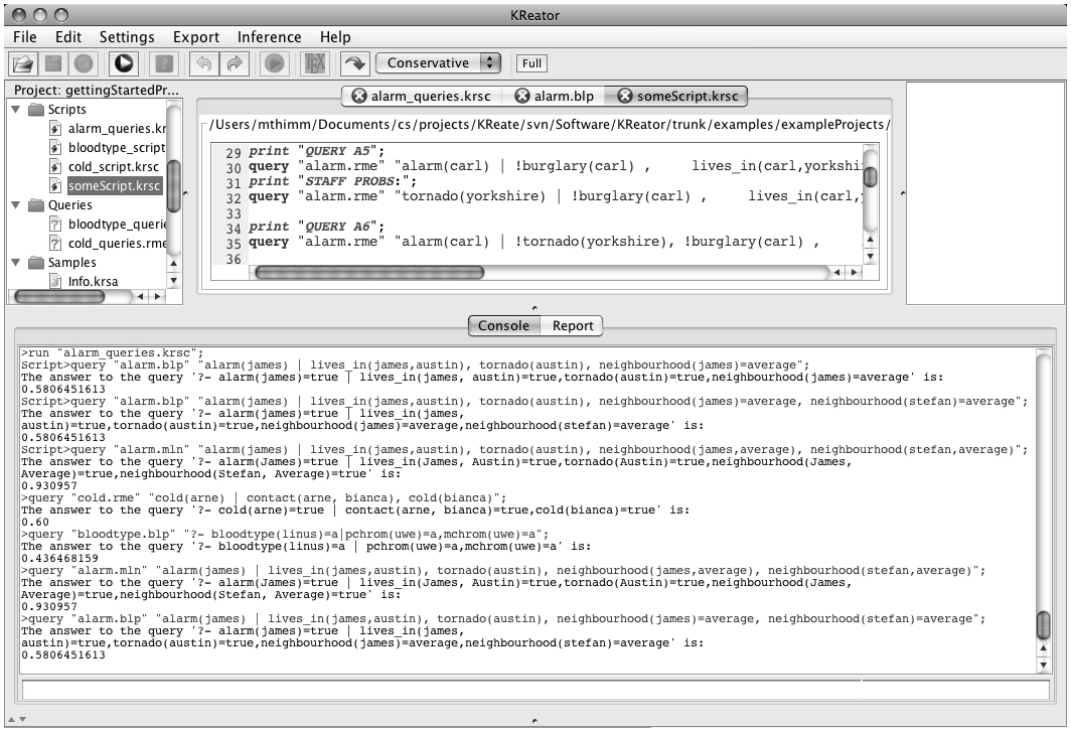

Fig. 3. The console with some KReatorScript

4.2 Querying a Knowledge Base

One of the most important tasks when working with knowledge bases is to address queries to a knowledge base, i. e. to infer knowledge. For that reason, KReator provides several functionalities which simplify the dealing with queries and make it more efficient.

KReator permits the processing of queries expressed in a *unified query syntax*. This query syntax abstracts from the respective syntax which is necessary to address a "native" query to a BLP, MLN, or RME knowledge base (and which also depends on the respective inference engine). That way, a query in unified syntax can be passed to an appropriate BLP, MLN, and RME knowledge base as well. The idea behind this functionality is, that some knowledge (cf. Ex. 1) can be modeled in different knowledge representation approaches (cf. Ex. 2, Ex. 4, and Ex. 5) and the user is able to compare these approaches in a more direct way. Such a comparison can then be done by formulating appropriate queries in unified syntax, passing them to the different knowledge bases, and finally analyzing the different answers, i. e. the probabilities. A KReator query in unified syntax consists of two parts: In the "head" of the query there are one or more ground atoms whose probabilities shall be determined. The "body" of the query is composed of several evidence atoms. For each supported knowledge representation formalism, KReator must convert a query in unified syntax in the exact syntax required by the respective inference engine. KReator also converts the respective output results to present them in a standardized format to the user. Figure 4 illustrates the processing of a query in unified syntax.

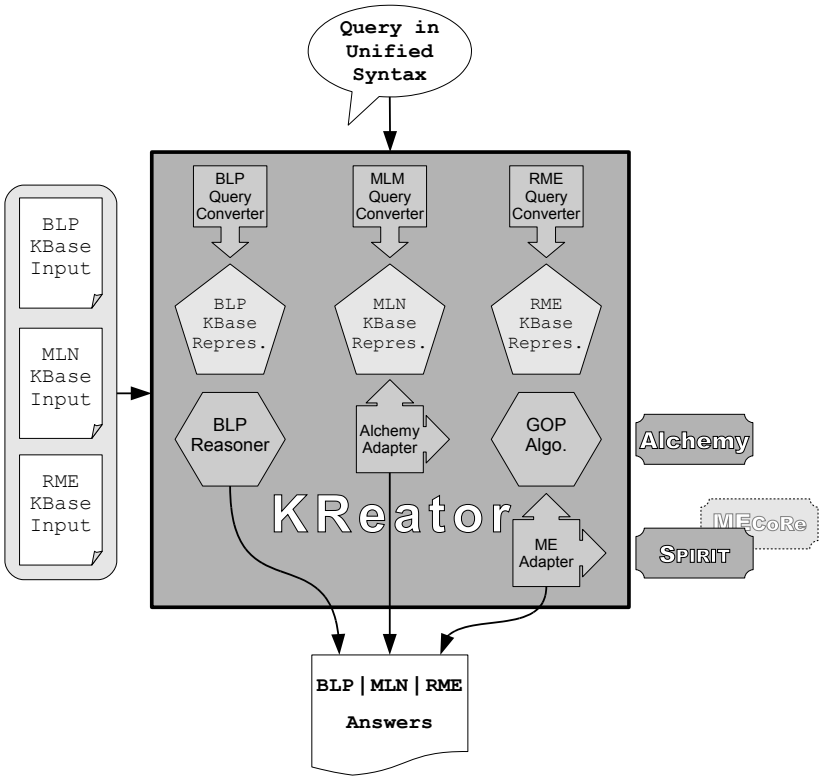

Fig. 4. Processing query in unified syntax

KReator offers the user an easy way to address a query to a knowledge base, simply by calling its query dialogue. In this dialogue (Fig. 5), the user can

input the atoms to be queried and he can conveniently specify the evidence. Since evidence usually consists of several atoms and is often reused for different queries, the user has the option to specify a file which contains the evidence to be to considered. While processing a query, the output area of the dialogue informs about important steps of the inference process. The calculated answer is clearly displayed in the lower part of the dialogue.

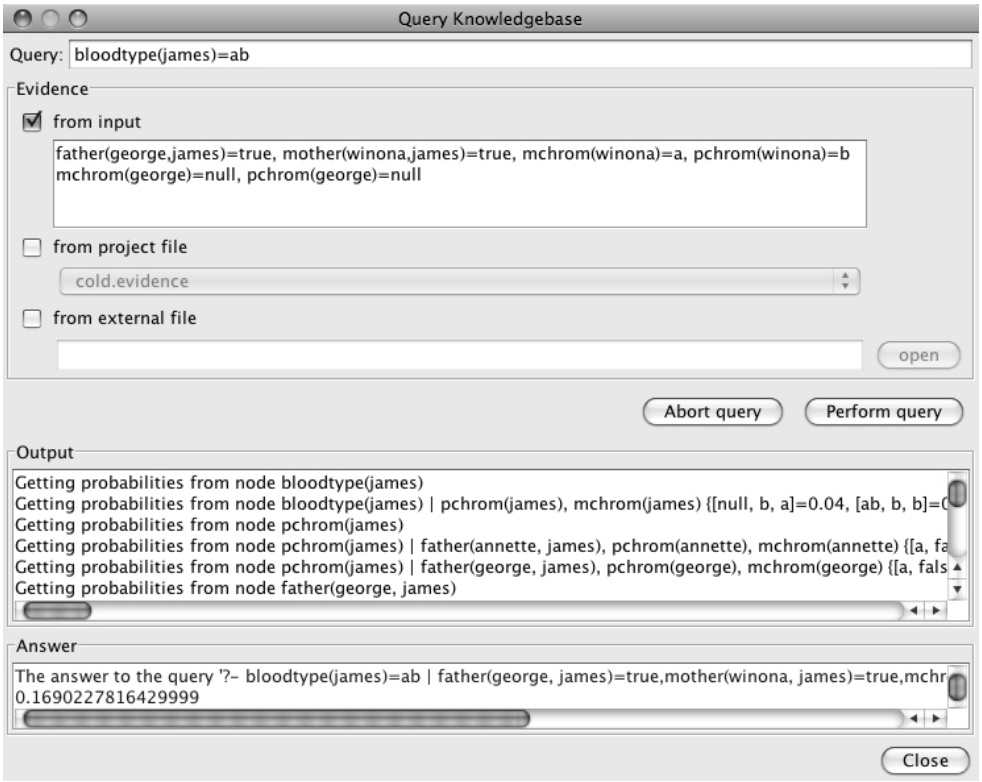

Fig. 5. Querying a knowledge base

Besides the capability of passing individual (i. e. ad-hoc) queries to a knowledge base, KREATOR also supports so-called *query collections*. A query collections is a file which contains several queries (either in unified or native syntax). Such a query collection can be passed to a knowledge base, so that all included queries are processed one after another. That way, KREATOR supports a persistent handling and batch-style processing of queries.

Example 6. Continuing our previously introduced burglary example consider the following evidence:

$$lives\_in(james, yorkshire), lives\_in(stefan, freiburg), burglary(james),$$
$$tornado(freiburg), neighborhood(james) = \mathsf{average}, neighborhood(stefan) = \mathsf{bad}$$

Table 1 shows three queries and their respective probabilities inferred from each of the example knowledge bases. The nine separate calculations altogether took

about three seconds. Each of the three knowledge bases represents Ex. 1 by a different knowledge representation approach. Nevertheless, the inferred probabilities are quite similar, except for the significant lower BLP probability of the query *alarm*(*stefan*). This results from using noisy-or as combing rule in the BLP calculations and from the fact that the CPDs of the BLP knowledge base carry some information not incorporated in the MLN and BLP knowledge bases.

	BLP	MLN	RME
alarm(*james*)	0.900	0.896	0.918
alarm(*stefan*)	0.650	0.896	0.907
burglary(*stefan*)	0.300	0.254	0.354

Table 1. Exemplary queries on different representations of the burglary example.

5 Summary and Future Work

In this paper we presented KReator and illustrated its system architecture and usage. Although KReator is still in an early stage of development it already supports Bayesian logic programs, Markov logic networks, and relational maximum entropy. Thus KReator is a versatile toolbox for probabilistic relational reasoning and alleviates the researcher's and knowledge engineer's work with different approaches to statistical relational learning.

Since KReator is still in an early development stage, there are a lot of plans on future development. The integration of adequate learning algorithms will be one of our major tasks in the near future, as our main focus so far was the integration of reasoning components. We also plan to integrate an extended version of the CONDOR system [18] and other formalisms for relational probabilistic knowledge representation such as logical Bayesian networks [7] and probabilistic relational models [9], as well as to use KReator as a testbed to evaluate other approaches for relational probabilistic reasoning under maximum entropy.

We plan to enhance KReator's unified query syntax to allow more complex queries. This requires more sophisticated conversion patterns to translate a unified query to the respective target syntax, e.g. to handle multi-state BLP predicates in an automated way. The enhancement of the query syntax will go along with the development of an even more challenging feature: We plan on introducing some kind of unified knowledge base (template) format. The final goal is to be able to formulate (at least) the central aspects of a knowledge base in a unified syntax and to have this knowledge base be converted to different target languages (at least semi-)automatically. Having this functionality available would dramatically improve the handling and comparison of different knowledge representation formalisms.

KReator is available under the GNU General Public License and can be obtained from `http://ls6-www.cs.uni-dortmund.de/kreator/`. The future development of KReator will also be strongly influenced be the feedback of early users. We will include such feedback in our development decisions and try to priorities such aspects which are most important to the users.

Acknowledgements. The research reported here was partially supported by the Deutsche Forschungsgemeinschaft (grants BE 1700/7-1 and KE 1413/2-1).

References

1. De Raedt, L., Kersting, K.: Probabilistic Inductive Logic Programming. In De Raedt, L., Kersting, K., Landwehr, N., Muggleton, S., Chen, J., eds.: Probabilistic Inductive Logic Programming. Springer (2008) 1–27
2. Cussens, J.: Logic-based Formalisms for Statistical Relational Learning. In Getoor, L., Taskar, B., eds.: An Introduction to Statistical Relational Learning. MIT Press (2007)
3. Kersting, K., De Raedt, L.: Bayesian Logic Programming: Theory and Tool. In Getoor, L., Taskar, B., eds.: An Introduction to Statistical Relational Learning. MIT Press (2007)
4. Domingos, P., Richardson, M.: Markov Logic: A Unifying Framework for Statistical Relational Learning. In: Proceedings of the ICML-2004 Workshop on Statistical Relational Learning and its Connections to Other Fields. (2004) 49–54
5. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1998)
6. Wellman, M.P., Breese, J.S., Goldman, R.P.: From Knowledge Bases to Decision Models. The Knowledge Engineering Review **7**(1) (1992) 35–53
7. Fierens, D.: Learning Directed Probabilistic Logical Models from Relational Data. PhD thesis, Katholieke Universiteit Leuven (2008)
8. Jaeger, M.: Relational Bayesian Networks: A Survey. Electronic Transactions in Artificial Intelligence **6** (2002)
9. Getoor, L., Friedman, N., Koller, D., Tasker, B.: Learning Probabilistic Models of Relational Structure. In Brodley, C.E., Danyluk, A.P., eds.: Proc. of the 18th International Conf. on Machine Learning (ICML 2001), Morgan Kaufmann (2001)
10. Baral, C., Gelfond, M., Rushton, N.: Probabilistic Reasoning with Answer Sets. Theory and Practice of Logic Programming (2009)
11. Loh, S.: Relational Probabilistic Inference Based on Maximum Entropy. Master's thesis, Technische Universität Dortmund (2009)
12. Thimm, M.: Representing Statistical Information and Degrees of Belief in First-Order Probabilistic Conditional Logic. In: Proceedings of the First Workshop on Relational Approaches to Knowledge Representation and Learning, Paderborn, Germany (September 2009)
13. Ketkar, N.S., Holder, L.B., Cook, D.J.: Comparison of graph-based and logic-based multi-relational data mining. SIGKDD Explor. Newsl. **7**(2) (2005) 64–71
14. Muggleton, S., Chen, J.: A Behavioral Comparison of some Probabilistic Logic Models. In Raedt, L.D., Kersting, K., Landwehr, N., Muggleton, S., Chen, J., eds.: Probabilistic Inductive Logic Programming. Springer (2008) 305–324
15. Kok, S., Singla, P., Richardson, M., Domingos, P., Sumner, M., Poon, H., Lowd, D., Wang, J.: The Alchemy System for Statistical Relational AI: User Manual. Department of Computer Science and Engineering, University of Washington. (2008)
16. Rödder, W., Meyer, C.H.: Coherent Knowledge Processing at Maximum Entropy by SPIRIT. In: Proceedings UAI 1996. (1996) 470–476
17. Finthammer, M., Beierle, C., Berger, B., Kern-Isberner, G.: Probabilistic Reasoning at Optimum Entropy with the MEcore System. In: Proceedings of FLAIRS'09, AAAI Press (2009)
18. Fisseler, J., Kern-Isberner, G., Beierle, C., Koch, A., Moeller, C.: Algebraic Knowledge Discovery using Haskell. In: Practical Aspects of Declarative Languages, 9th International Symposium. Springer (2007)

Representing Statistical Information and Degrees of Belief in First-Order Probabilistic Conditional Logic

Matthias Thimm

Information Engineering Group, Department of Computer Science,
Technische Universität Dortmund, Germany

Abstract. Employing maximum entropy methods on probabilistic conditional logic has proven to be a useful approach for commonsense reasoning. Yet, the expressive power of this logic and similar formalisms is limited due to their foundations on propositional logic and in the past few years a lot of proposals have been made for probabilistic reasoning in relational settings. Most of these proposals rely on extensions of traditional graph-based probabilistic models like Bayes nets or Markov nets whereas probabilistic conditional logic does not presuppose any graphical structure underlying the model to be represented. In this paper we take an approach of lifting maximum entropy methods to the relational case by using a first-order version of probabilistic conditional logic. Furthermore, we take a specific focus on representing relational probabilistic knowledge by differentiating between different intuitions on relational probabilistic conditionals, namely between statistical interpretations and interpretations on degrees of belief. We develop a list of desirable properties on an inference procedure that supports these different interpretations and propose a specific inference procedure that fulfills these properties. We furthermore discuss related work and give some hints on future research.

1 Introduction

Applying probabilistic reasoning to relational representations of knowledge is a very active and controversy research area. In the past few years the fields of *probabilistic inductive logic programming* and *statistical relational learning* put forth a lot of proposals that deal with combining traditional probabilistic models of knowledge like Bayes nets or Markov nets [1] with first-order logic, see [2, 3] for some excellent surveys. For example, two of the most prominent approaches for extending propositional approaches to the relational case are Bayesian logic programs [4] and Markov logic networks [5]. While Bayesian logic programs extend Bayes nets using a logic programming language Markov logic networks extend Markov nets using a restricted form of first-order logic. Both frameworks use knowledge-based model construction techniques [6, 7] to reduce the problem of probabilistic reasoning in a relational context to probabilistic reasoning in a propositional context. In both frameworks—and also in most other approaches—this is done by appropriately grounding the parts of the knowledge base that are

needed for answering a particular query and treating this grounded parts as a propositional knowledge base. While most approaches to relational probabilistic reasoning employ graphical models for probabilistic reasoning, in this paper we take another direction by lifting probabilistic conditional logic [8, 9] to the first-order case and applying maximum entropy methods [10–12] for reasoning.

In (propositional) probabilistic conditional logic knowledge is captured using conditionals of the form $(\phi \mid \psi)[\alpha]$ with some formulas ϕ, ψ of a given propositional language and $\alpha \in [0, 1]$. A probabilistic conditional of this form partially describes an (unknown) probability distribution P^* by stating that $P^*(\phi \mid \psi) = \alpha$ holds. In contrast to Bayes nets probabilistic conditional logic does not demand to fully describe a probability distribution but only to state constraints on it. On the one hand this is of great advantage because normally the knowledge engineer cannot fully specify a probability distribution for the problem area at hand. For example, if one has to represent probabilistic information on the relationships between symptoms and diseases then (usually) one can specify the probability of a specific disease given that a specific symptom is present but not if the symptom is not present. Probabilistic conditional logic avoids such problems by allowing to only partially specify a probability distribution. On the other hand, an incomplete specification of the problem area may lead to inconclusive inferences because there may be multiple probability distributions that satisfy the specified knowledge. The naïve approach to reason in probabilistic conditional logic is to compute upper and lower bounds for specific queries by consulting every probability distribution that is a model of the given knowledge base. While this skeptical form of reasoning may be appropriate for some applications, usually the inferences of this approach tend to be too weak to be meaningful. As a credulous alternative, one can select a specific probability distribution from the models of the knowledge base and do reasoning by just using this probability distribution. A reasonable choice for such a model is the one probability distribution with maximum entropy [10–12]. This probability distribution satisfies several desirable properties for commonsense reasoning and is uniquely determined among the probability distributions that satisfy a given set of probabilistic conditionals, see [10–12] for the theoretical foundations.

While applying maximum entropy methods in a direct fashion onto a first-order probabilistic conditional logic has already been investigated for example in [13–15], in this paper we take a special focus on relational probabilistic knowledge representation, namely the differentiation between statistical information and degrees of belief [16–18]. When considering conditionals, the modeled knowledge becomes ambiguous by introducing variables. Consider the following example inspired by [19]:

$$(likes(X, Y) \mid elephant(X) \wedge keeper(Y))[0.8]$$
$$(likes(X, fred) \mid elephant(X) \wedge keeper(fred))[0.4]$$
$$(likes(clyde, fred) \mid elephant(clyde) \wedge keeper(fred))[0.6]$$

The first conditional represents the information that with a probability of 0.8 a (typical) elephant likes his (typical) keeper. The second conditional states that

a (typical) elephant likes the keeper $fred$ with a probability of 0.4 and the third conditional states that the elephant *clyde* likes the keeper $fred$ with a probability of 0.6. From a commonsensical point of view this knowledge base makes perfect sense. Given an adequate population of elephants and keepers this knowledge base says that typically an elephant likes his keeper, $fred$ is an exception and mostly unpopular, but *clyde* likes $fred$ still a bit more. But when treating the first two conditionals as schemas for their propositional instantiations (given a finite universe) then the grounded knowledge base becomes inconsistent because there are instantiations of the first two conditionals that are in direct conflict with each other and with the third conditional. The problem of inconsistency arises when treating conditionals like the first one as schemas for conditionals on the degrees of belief. But presumably what one really want to model when representing conditionals of this form is some kind of statistical information or maybe a default rule [19]. In the example above, the first conditional describes some form of statistical distribution on the all pairs of elephants and keepers and the second conditional describes a distribution on all elephants. In contrast to the first two conditionals the third conditional does not mention any variable. In fact, it mentions only ground instances regarding the constants *clyde* and $fred$ thus describing a *degree of belief* on the truth-value of $likes(clyde, fred)$ given that *clyde* is an elephant and $fred$ is a keeper. As a consequence, the knowledge represented by the third conditional describes some belief on the distribution of possible worlds rather than on the individuals of the universe. In this paper, we argue that an explicit differentiation of this two types of knowledge is important in order to reason with relational probabilistic knowledge bases.

The rest of this paper is organized as follows. In the following Section 2 we give a brief overview on (propositional) probabilistic conditional logic and continue in Section 3 with syntax and semantics of its extension to the first-order case. In Section 4 we propose some properties a reasonable inference mechanism should fulfill in order to interpret a relational probabilistic knowledge base in the sense described above and present our approach for an inference mechanism afterwards. In Section 5 we discuss our approach and review related work. In Section 6 we conclude.

2 Preliminaries

Before introducing probabilistic conditional logic for a relational language we begin by giving an overview on (propositional) probabilistic conditional logic. We extend this framework to the relational case in the subsequent section. But first, we consider a framework of propositional variables. Let $\mathcal{V} = \{V_1, \ldots, V_n\}$ be a set of propositional variables with finite domains $\mathsf{Dom}(V_1), \ldots, \mathsf{Dom}(V_n)$. An expression of the form $V_i = v_i$ is called a *literal* if v_i is in the domain of V_i, i. e. $v_i \in \mathsf{Dom}(V_i)$. The language $\mathcal{L}_\mathcal{V}$ is generated using the connectives $\neg$, $\wedge$, and $\vee$ on the literals in $\mathcal{V}$ in the usual way. For arbitrary formulas ϕ, ψ we abbreviate conjunctions $\phi \wedge \psi$ by $\phi\psi$ and negation $\neg\phi$ by overlining $\overline{\phi}$. If V is a binary variable, i. e., it is $\mathsf{Dom}(V) = \{\mathsf{true}, \mathsf{false}\}$, we abbreviate $V = \mathsf{true}$ by just V and

$V = \mathsf{false}$ by $\overline{V}$. We write $\top$ for tautological formulas, e. g. $\phi \vee \overline{\phi} \equiv \top$. A *possible world* (interpretation) assigns to each variable $V_i \in \mathcal{V}$ a value in $\mathsf{Dom}(V_i)$. If ω is a possible world, then $\omega \models (V_i = v_i)$ if and only if ω assigns v_i to V_i. For an arbitrary formula ϕ the expression $\omega \models \phi$ evaluates in the usual way. Let $\Omega_{\mathcal{V}}$ be the set of all possible worlds of $\mathcal{L}_{\mathcal{V}}$.

Propositional probabilistic knowledge bases are build using propositional probabilistic conditionals, that impose certain restrictions on the conditional probabilities of the models of the knowledge base. A *(propositional) probabilistic conditional* r is an expression of the form $r = (\phi \,|\, \psi)[\alpha]$ with formulas ϕ, ψ and $\alpha \in [0, 1]$. If $\psi \equiv \top$ we write $(\phi)[\alpha]$ instead of $(\phi \,|\, \top)[\alpha]$. A set of probabilistic conditionals $R = \{r_1, \ldots, r_m\}$ is called a *(propositional) knowledge base*. The models of a knowledge base R are the probability distributions $P : \Omega_{\mathcal{V}} \to [0, 1]$ that fulfill all restrictions on the conditional probabilities imposed by the probabilistic conditionals in R. More specifically, a probability distribution $P : \Omega_{\mathcal{V}} \to [0, 1]$ is a model for a knowledge base R, written $P \models R$, if and only if $P \models r$ for every $r \in R$. That is

$$\begin{aligned} P \models (\phi \,|\, \psi)[\alpha] \;&:\Longleftrightarrow\; P(\phi \,|\, \psi) = \alpha \text{ and } P(\psi) > 0 \\ &\Longleftrightarrow\; \frac{P(\phi\psi)}{P(\psi)} = \alpha \text{ and } P(\psi) > 0 \end{aligned}$$

with

$$P(\phi) = \sum_{\omega \in \Omega_{\mathcal{V}}, \omega \models \phi} P_R(\omega) \quad .$$

A knowledge base R made of probabilistic conditionals describes incomplete knowledge. Usually, one is interested in performing inductive representation techniques and thus in computing a single probability distribution that describes R best and gives a complete description of the problem area at hand. This can be done using methods based on maximum entropy, which feature several nice properties [11, 10, 12, 20]. The entropy $H(P)$ of a probability distribution P is defined as

$$H(P) = -\sum_{\omega \in \Omega_{\mathcal{V}}} P(\omega) \log P(\omega)$$

and measures the amount of indeterminateness inherent in P. By selecting the probability distribution P^* among all probability distributions that satisfy a given knowledge base R, i. e. by computing the solution to the optimization problem

$$P^* := \mathsf{ME}(R) = \arg\max_{P \models R} H(P) \quad ,$$

we get the one probability distribution that satisfies R and adds as little information as necessary.

3 Syntax and Semantics of First-Order Conditional Logic

In the following we give an extension of probabilistic conditional logic to the relational case similar as in [21, 14]. To simplify presentation we use the same

names for logical constructs as in propositional conditional language, e. g. we will refer to *relational probabilistic conditionals* just by *probabilistic conditionals*.

Let $\mathcal{L}_D$ be a first-order language with a fixed finite universe (domain) D without quantifiers and functions. We denote variables with a beginning uppercase, constants with a beginning lowercase letter, and vectors of these in boldface. We use greek letters ϕ and ψ for formulas and ξ for sentences (formulas with no free variables). For a first-order formula ϕ let $\text{FREE}(\phi)$ denote the set of (free) variables appearing in ϕ. We will write $\phi(\boldsymbol{X})$ to explicitly name the variables $\boldsymbol{X}$ in ϕ and we denote by $\phi(\boldsymbol{c})$ the grounded instance of ϕ with constants $\boldsymbol{c}$. Let $\mathsf{ground}_C(\phi)$ denote the set of grounded instances of ϕ with respect to a set of constants C.

Definition 1 (Probabilistic Conditional). *An expression of the form*

$$(\phi \,|\, \psi)[\alpha]$$

with first-order formulas ϕ, ψ (not necessarily ground) and a real $\alpha \in [0, 1]$ is called a probabilistic conditional*. A probabilistic conditional $(\phi \,|\, \psi)[\alpha]$ is* ground *if $\text{FREE}(\phi) = \text{FREE}(\psi) = \emptyset$.*

As above we abbreviate $(\phi \,|\, \top)[\alpha]$ by $(\phi)[\alpha]$. For a probabilistic conditional $(\phi \,|\, \psi)[\alpha]$ let $\mathsf{ground}_C((\phi \,|\, \psi)[\alpha])$ denote the set of all grounded probabilistic conditionals of $(\phi \,|\, \psi)[\alpha]$ with respect to the set of constants C.

Definition 2 (Knowledge base). *A finite set R of probabilistic conditionals is called a* knowledge base*. A knowledge base R is* ground *if every probabilistic conditional in R is ground. Let $\mathfrak{R}$ denote the set of knowledge bases and $\mathfrak{R}_P \subseteq \mathfrak{R}$ the set of ground knowledge bases.*

Remark 1. Bear in mind that a ground knowledge base $R \in \mathfrak{R}_P$ is equivalent to a propositional knowledge base R' by interpreting ground atoms in R as ordinary propositional atoms. For the rest of this paper we treat ground relational knowledge bases and propositional knowledge bases interchangeably.

The informal semantics of a probabilistic conditional $(\phi \,|\, \psi)[\alpha]$ are as follows.

- If $\text{FREE}(\phi\psi) = \emptyset$ we interpret $(\phi \,|\, \psi)[\alpha]$ as an *uncertainty assessment* over the possible worlds as in propositional probabilistic conditional logic, thus specifying a degree of belief on a conditional probability.
- If $\text{FREE}(\phi\psi) \neq \emptyset$ we interpret $(\phi \,|\, \psi)[\alpha]$ as a *statistical assessment* stating that in the actual world a portion α of all ψ's are ϕ's.

We illustrate our intuition behind this informal semantics by means of an example.

Example 1. Consider again the scenario from the introduction. Let R be a knowledge base given as follows.

$$\begin{aligned} R = \quad \{ \quad & (likes(X, Y) \,|\, elephant(X) \wedge keeper(Y))[0.8] \\ & (likes(X, fred) \,|\, elephant(X) \wedge keeper(fred))[0.4] \\ & (likes(clyde, fred) \,|\, elephant(clyde) \wedge keeper(fred))[0.6] \quad \} \quad . \end{aligned}$$

In R we can assign the following informal meanings to the individual probabilistic conditionals:

- $(likes(clyde, fred) \,|\, elephant(clyde) \wedge keeper(fred))[0.6]$
  This conditional states that our subjective degree of belief of *clyde* liking $fred$ is 0.6. So, if we know that *clyde* is an elephant and $fred$ is a keeper we expect in 60 % of all occasions that *clyde* likes $fred$.
- $(likes(X, fred) \,|\, elephant(X) \wedge keeper(fred))[0.4]X$
  This conditional states that we expect 40 % of all elephants to like $fred$.
- $(likes(X, Y) \,|\, elephant(X) \wedge keeper(Y))[0.8]X, Y$
  This conditional states that we expect for 80 % of all elephant-keeper combinations that the elephant likes the keeper.

For a knowledge base R we denote by $\mathsf{Bel}(R)$ its projection on $\mathfrak{R}_P$, i. e., it is $\mathsf{Bel}(R) = \{r \in R \,|\, \text{FREE}(r) = \emptyset\}$. In other words, $\mathsf{Bel}(R)$ contains all uncertainty assessments. Analogously, let $\mathsf{Stat}(R) = R \backslash \mathsf{Bel}(R)$ denote the set of all statistical assessments of R.

Formal semantics for first-order conditional logic are given by probability distributions. The probability distributions under consideration are defined over the possible worlds of the given first-order language $\mathcal{L}_D$. A possible world ω for $\mathcal{L}$ is a tuple $\omega = \langle D, I \rangle$ with domain D and interpretation I which maps in the usual way constants to domain elements, unary predicate symbols to subsets of D and so on. As a simplification we interpret constants by themselves, i. e., for any constant c it is $I(c) = c$ in any possible world ω. As D is fixed for all possible worlds we will identify $\omega = \langle D, I \rangle$ with I when appropriate. Let Ω_D be the set of all these possible worlds with domain D and so we are interested in probability distributions $P : \Omega_D \to [0, 1]$. Let Prob_D be the set of probability distributions for domain D. $P \in \mathsf{Prob}_D$ is extended on first-order sentences (ground formulas) ξ by

$$P(\xi) = \sum_{\omega \models \xi} P(\omega) \quad .$$

Interpreting uncertainty assessments with probability distributions can be done analogously like in the propositional case. The problem at hand arises when considering statistical assessments like

$$c = (likes(X, fred) \,|\, elephant(X) \wedge keeper(fred))[0.5] \quad .$$

What is an appropriate satisfaction relation $\models^{cp}$ such that for a probability distribution P the statement $P \models^{cp} c$ describes our intuition on statistical assessments described above? We propose a new satisfaction relation $\models^{cp}_D$ on probabilistic conditionals that specifies when a probability distribution $P \in \mathsf{Prob}_D$ satisfies a given probabilistic conditional r. For the case of an uncertainty assessment $(\phi(\boldsymbol{c}) \mid \psi(\boldsymbol{c}))$, we define the satisfaction relation $\models^{cp}_D$ through

$$P \models^{cp}_D (\phi(\boldsymbol{c}) \,|\, \psi(\boldsymbol{c}))[\alpha] \quad :\Leftrightarrow \quad P((\phi(\boldsymbol{c}) \mid \psi(\boldsymbol{c}))) = \alpha \tag{1}$$

as in the propositional case. For a statistical assessment $(\phi(\boldsymbol{X}) \,|\, \psi(\boldsymbol{X}))[\alpha]$, we say that a probability distribution P satisfies $(\phi(\boldsymbol{X}) \,|\, \psi(\boldsymbol{X}))[\alpha]$ if the average of

the conditional probabilities of all instantiations of $(\phi(\boldsymbol{X}) \mid \psi(\boldsymbol{X}))[\alpha]$ is α. So it is $P \models^{cp}_D (\phi(\boldsymbol{X}) \mid \psi(\boldsymbol{X}))[\alpha]$ if and only if

$$\frac{\sum_{(\phi(\boldsymbol{c})|\psi(\boldsymbol{c}))\in \mathsf{ground}_D((\phi(\boldsymbol{X})|\psi(\boldsymbol{X})))} P(\phi(\boldsymbol{c}) \mid \psi(\boldsymbol{c}))}{|\mathsf{ground}_D(\phi(\boldsymbol{X}) \mid \psi(\boldsymbol{X}))|} = \alpha \quad . \tag{2}$$

Notice, that Equation (2) also subsumes the case of uncertainty assessments in Equation (1) as a special case. As usual, a probability distribution P satisfies a knowledge base R, denoted $P \models^{cp}_D R$, if P satisfies every probabilistic conditional $r \in R$. We say that R is consistent iff there is at least on P with $P \models^{cp}_D R$, otherwise R is inconsistent.

4 Inference in First-Order Conditional Logic

We are interested in finding a "good" probability distribution P that satisfies all probabilistic conditionals of a given knowledge base R. More specifically, we are interested in a function $\mathsf{SRME}(R)$ (Statistical relational maximum entropy) that takes a knowledge base R and gives a probability distribution $P = \mathsf{SRME}(R)$ as output such that P describes R "best" in a commonsensical manner. In the following we state some properties on the operator SRME that derive from our intuition and afterwards describe such a function that fulfills these properties.

4.1 Desirable Properties

When considering knowledge bases like the one in Example 1 we want to be able to name a single probability distribution P that is the "best" model of R. Taking a naïve approach by grounding all conditionals in R universally and taking this grounding R' as a propositional knowledge base, we can not determine any probability distribution that satisfies R' due to its inherent inconsistency [15]. So our first demand on an appropriate operator SRME is its well-definedness. In the following, let $\mathsf{SRME} : \mathfrak{R} \to \mathsf{Prob}_D$ be an operator that maps a knowledge base $R \in \mathfrak{R}$ onto a probability distribution $P = \mathsf{SRME}(R) \in \mathsf{Prob}_D$ such that P commonsensical describes R.

(Well-Definedness) If R is a consistent then $\mathsf{SRME}(R)$ is well-defined.

We need some further notation to go on. For a formula ϕ let $\phi[d/c]$ denote the formula that is the same as ϕ except that every occurrence of the term c (either a variable or a constant) is substituted with the term d. More generally, let $\phi[d_1/c_1, \ldots, d_n/c_n]$ denote the formula that is the same as ϕ except that every occurrence of c_i is substituted with d_i for $1 \leq i \leq n$ simultaneously. The substitution operator $[\cdot]$ is extended on sets of formulas and conditionals in the usual way.

When considering knowledge bases based on a relational language the beliefs one gains on specific individuals is of special interest. An important demand to made is that the information one gains for different individuals is the same

when these individuals are indistinguishable. More specifically, if the explicit information encoded in R for two different individuals $c_1, c_2 \in D$ is the same the probability distribution P should treat them indistinguishable. We can formalizing this indistinguishable property by introducing an equivalence relation on constants.

Definition 3 (Syntactical Equivalence). *Let R be a knowledge base. The constants $c_1, c_2 \in D$ are* syntactical equivalent*, denoted by $c_1 \equiv_R c_2$, if and only if $R = R[a_1/a_2, a_2/a_1]$.*

Observe that $\equiv_R$ is indeed an equivalence relation, i. e., it is reflexive, transitive, and symmetric. The equivalence classes of $\equiv_R$ are called *R-equivalence classes* and the set of all R-equivalence classes is denoted by $\mathcal{S}_R$. Note, that the notion of syntactical equivalence bears a resemblance with the notion of *reference classes* [17] but on a pure syntactical level.

Using syntactical equivalence we can state our demand for equal treatment of indistinguishable individuals as follows.

(Prototypical Indifference) Let R be a knowledge base and ξ a ground sentence. For any $c_1, c_2 \in D$ with $c_1 \equiv_R c_2$ it is

$$\mathsf{SRME}(R)(\xi) = \mathsf{SRME}(R)(\xi[c_1/c_2, c_2/c_1]) \quad .$$

From (Prototypical Indifference) some generalizations follow naturally.

Proposition 1. *Let* SRME *satisfy (Prototypical Indifference).*

1. *Let R be a knowledge base and ξ_1, ξ_2 be two ground sentences. For $c_1, c_2 \in D$ with $c_1 \equiv_R c_2$ it holds*

$$\mathsf{SRME}(R)(\xi_1 \mid \xi_2) = \mathsf{SRME}(R)(\xi_1[c_1/c_2, c_2/c_1] \mid \xi_2[c_1/c_2, c_2/c_1]) \quad .$$

2. *Let $S \in \mathcal{S}_R$, $c_1, \ldots, c_n \in S$, and $\sigma : S \to S$ a permutation on S, i. e. a bijective function on S. Then it holds for a ground sentence ξ*

$$\mathsf{SRME}(R)(\xi) = \mathsf{SRME}(R)(\xi[\sigma(c_1)/c_1, \ldots, \sigma(c_n)/c_n]) \quad .$$

Proof.

1. *Because of (Prototypical Indifference) it holds directly*

$$\begin{aligned} \mathsf{SRME}(R)(\xi_2) &= \mathsf{SRME}(R)(\xi_2[c_1/c_2, c_2/c_1]) \quad \textit{and} \\ \mathsf{SRME}(R)(\xi_1 \wedge \xi_2) &= \mathsf{SRME}(R)((\xi_1 \wedge \xi_2)[c_1/c_2, c_2/c_1]) \end{aligned}$$

and hence

$$\begin{aligned} \mathsf{SRME}(R)(\xi_1 \mid \xi_2) &= \frac{\mathsf{SRME}(R)(\xi_1 \wedge \xi_2)}{\mathsf{SRME}(R)(\xi_2)} \\ &= \frac{\mathsf{SRME}(R)(\xi_1 \wedge \xi_2[c_1/c_2, c_2/c_1])}{\mathsf{SRME}(R)(\xi_2[c_1/c_2, c_2/c_1])} \\ &= \mathsf{SRME}(R)(\xi_1[c_1/c_2, c_2/c_1] \mid \xi_2[c_1/c_2, c_2/c_1]) \end{aligned}$$

due to $(\xi_1 \wedge \xi_2)[x_i/y_i]_{i=1,\ldots,n} = \xi_1[x_i/y_i]_{i=1,\ldots,n} \wedge \xi_2[x_i/y_i]_{i=1,\ldots,n}$.

2. *This follows from the fact that every permutation can be represented as a product of transpositions [22], i. e. permutations that exactly transpose two elements. Let* $\sigma_1, \ldots, \sigma_m$ *be these transpositions of* σ *and let* $\sigma_{1\ldots i} = \sigma_i \circ \ldots \circ \sigma_1$ *for* $i = 1, \ldots, m$*. Note, that* $\sigma_{1\ldots 1} = \sigma_1$ *and* $\sigma_{1\ldots m} = \sigma$*. Due to (Prototypical Indifference) it holds*

$$\mathsf{SRME}(R)(\xi) = \mathsf{SRME}(R)(\xi[\sigma_1(c_1)/c_1, \ldots, \sigma_1(c_n)/c_n])$$

and for any $i = 2 \ldots, m$ *it holds*

$$\begin{aligned} &\mathsf{SRME}(R)(\xi[\sigma_{1\ldots i-1}(c_1)/c_1, \ldots, \sigma_{1\ldots i-1}(c_n)/c_n]) \\ &= \mathsf{SRME}(R)(\xi[\sigma_{1\ldots i}(c_1)/c_1, \ldots, \sigma_{1\ldots i}(c_n)/c_n]) \quad . \end{aligned}$$

Via transitivity and $\sigma_{1\ldots m} = \sigma$ *it follows*

$$\mathsf{SRME}(R)(\xi) = \mathsf{SRME}(R)(\xi[\sigma(c_1)/c_1, \ldots, \sigma(c_n)/c_n]) \quad .$$

□

Another aspect that should be satisfied by the operation SRME is some form of compatibility to the propositional case. For (relational) knowledge bases that are equivalent to propositional knowledge bases, i. e., ground knowledge bases, the operation SRME should coincide with the ME operator on propositional knowledge bases, cf. Section 2.

(Compatibility I) Let R be a ground knowledge base. If ξ is a ground sentence then it is $\mathsf{ME}(R)(\xi) = \mathsf{SRME}(R)(\xi)$.

Moreover, as the uncertainty assessments of a knowledge base R describe "strict" uncertain knowledge the probability distribution $\mathsf{SRME}(R)$ should reflect this knowledge faithfully.

(Compatibility II) Let R be a knowledge base. If $(\phi(\boldsymbol{c}) \,|\, \psi(\boldsymbol{c}))[\alpha] \in R$ is an uncertainty assessment it is

$$\mathsf{SRME}(R)(\phi(\boldsymbol{c}) \mid \psi(\boldsymbol{c})) = \alpha \quad .$$

So far, we have not taken into account the intention for representing statistical assessments. Given a statistical assessment $r = (\phi(\boldsymbol{X}) \,|\, \psi(\boldsymbol{X}))[\alpha]$ our intention in representing r in a knowledge base R is that for every instantiation $r' = (\phi(\boldsymbol{c}) \,|\, \psi(\boldsymbol{c}))[\alpha]$ of r the conditional probability of $\phi(\boldsymbol{c})$ given $\psi(\boldsymbol{c})$ "should" be α. But how do we capture this intention? Surely, we cannot guarantee that every possible instantiation r' of r will conform to a strict interpretation of this demand. This follows mainly from the fact, that using uncertainty assessments we should be able to give exceptions to this rule, cf. Example 1. What we are really want to describe when representing a statistical assessment r is that *given an adequate large domain* the conditional probability of the bigger part of the interpretations (neglecting exceptions) will converge towards α. This behavior resembles the intuition behind the "Law of Large Numbers" [23].

(Convergence) Let $R_1, R_2, \ldots$ be knowledge bases on $\mathcal{L}_{D_1}, \mathcal{L}_{D_2}, \ldots$ with $R_1 = R_2 = \ldots$ and $D_1 \subset D_2 \subset \ldots$ (for $i \in \mathbb{N}^+$). For a statistical assessment $r = (\phi(\boldsymbol{X}) \,|\, \psi(\boldsymbol{X}))[\alpha] \in R_1$ let $r' = (\phi(\boldsymbol{c}) \,|\, \psi(\boldsymbol{c}))[\alpha]$ be a proper instantiation of r with constants $\boldsymbol{c}$ that do not appear in R_1. For any such r and r' it is

$$\lim_{i \to \infty} \mathsf{SRME}(R_i)(\phi(\boldsymbol{c}) \mid \psi(\boldsymbol{c})) = \alpha$$

Another aspect of statistical assessments is their capability to comprehend for exceptions. Usually, statistical assessments are defined to model some kind of *expected value* over the set of instantiations. As such, if the probability of one instantiation of a statistical assessment lies below the value of the statement there has to be another instantiation with a probability higher than the value of the statement in order to compensate for the other exception (remember that the domain D is assumed to be finite).

(Compensation) Let R be a knowledge base and $(\phi(\boldsymbol{X}) \,|\, \psi(\boldsymbol{X}))[\alpha] \in R$ a statistical assessment with $\alpha \in (0, 1)$ (the open interval). If $\boldsymbol{c_1}$ is a vector of constants such that $\mathsf{SRME}(R)(\phi(\boldsymbol{c_1}) \mid \psi(\boldsymbol{c_1})) < \alpha$ then there is another vector of constants $\boldsymbol{c_2}$ with $\mathsf{SRME}(R)(\phi(\boldsymbol{c_2}) \mid \psi(\boldsymbol{c_2})) > \alpha$.

4.2 Statistical Relational Maximum Entropy

In the following we define a function $\mathsf{SRME}_1 : \mathfrak{R} \to \mathsf{Prob}_D$ that fulfills the desired properties defined in the previous section. We define the function $\mathsf{SRME}_1(R)$ using the the proposed semantics $\models_D^{cp}$ analogously like in the propositional case by selecting a probability distribution with maximum entropy among all probability distributions that satisfy R.

$$\mathsf{SRME}_1(R) = \arg \max_{P \models_D^{cp} R} - \sum_{\omega \in \Omega} P(\omega) \log P(\omega) \tag{3}$$

Remark 2. Note, that it seems that the optimization problem defined by Equation (3) is (in general) not uniquely solvable because the set of probability distributions defined by Equation (2) is non-convex. However, preliminary experimental results indicate that the probability distribution in Equation (3) is uniquely determined. As the formal proof has yet to be made, we assume for the rest of this paper that in Equation (3) an arbitrary probability distribution with maximum entropy will be chosen. Nonetheless, if R is consistent there is at least one probability distribution with maximum entropy that can be chosen in Equation (3).

Example 2. We continue Example 1. Let $\mathcal{L}_D$ be a first-order language with predicates $elephant/1$, $keeper/1$, and $likes/2$ and domain $D = \{clyde, dumbo,$

$catty, giddy, fred, dave\}$. Let R be given by

$$(elephant(clyde))[1] \tag{4}$$
$$(elephant(dumbo))[1] \tag{5}$$
$$(elephant(catty))[1] \tag{6}$$
$$(elephant(giddy))[1] \tag{7}$$
$$(keeper(fred))[1] \tag{8}$$
$$(keeper(dave))[1] \tag{9}$$
$$(likes(X,Y) \,|\, elephant(X) \wedge keeper(Y))[0.5] \tag{10}$$
$$(likes(X,fred) \,|\, elephant(X) \wedge keeper(fred))[0.25] \tag{11}$$
$$(likes(clyde,fred) \,|\, elephant(clyde) \wedge keeper(fred))[0.1] \tag{12}$$

Here, the conditional (10) states that in 50 % of the elephant/keeper combinations the elephant likes the keeper, conditional (11) states, that 25 % of the elephants like $fred$ and conditional (12) states that the probability of *clyde* liking $fred$ is 0.1. In the following we give the probabilities of several instantiations of *likes* in $\mathsf{SRME}_1(R)$. Notice, how the probabilities of the instantiations of the conditionals (10) and (11) change in order to compensate for the exceptional instantiations involving *clyde* and $fred$.

$$\begin{aligned}
\mathsf{SRME}_1(R)(likes(clyde,dave)) &= 0.75\\
\mathsf{SRME}_1(R)(likes(dumbo,dave)) &= 0.75\\
\mathsf{SRME}_1(R)(likes(catty,dave)) &= 0.75\\
\mathsf{SRME}_1(R)(likes(giddy,dave)) &= 0.75\\
\mathsf{SRME}_1(R)(likes(clyde,fred)) &= 0.1\\
\mathsf{SRME}_1(R)(likes(dumbo,fred)) &= 0.3\\
\mathsf{SRME}_1(R)(likes(catty,fred) &= 0.3\\
\mathsf{SRME}_1(R)(likes(giddy,fred)) &= 0.3
\end{aligned}$$

As there is no additional information on the elephants in R except *clyde* they have to be treated in the same manner. Due to conditional (11) every elephant is equally likely to like $fred$ with a probability 0.3 and due to conditional (12) *clyde* likes fred with probability 0.1. Due to conditional (10) the total percentage of *like* relations that hold have to be 50 %. This information increases the probability of the elephants liking *dave* accordingly to 75 %.

Considering the comments in Remark 2 we first state the following conjecture.

Conjecture 1. SRME_1 satisfies (Well-Definedness).

In the following we give some theoretical results mostly in form of proof sketches that show that the proposed operator SRME_1 indeed fulfills the desired properties discussed in Section 4.1.

Proposition 2. SRME_1 *satisfies (Prototypical Indifference).*

Proof. (Sketch) This is ensured by selecting in Equation (3) a probability distribution with maximum entropy. Suppose $(\phi(\boldsymbol{X}) \mid \psi(\boldsymbol{X}))[\alpha] \in R$ *is a statistical assessment,* $\boldsymbol{c_1}, \boldsymbol{c_2}$ *vectors of constants that only differ in constants that do not appear in* R*. If* $p_1 = \mathsf{SRME}_1(R)(\phi(\boldsymbol{c_1}) \mid \psi(\boldsymbol{c_1})) \neq \mathsf{SRME}_1(R)(\phi(\boldsymbol{c_2}) \mid \psi(\boldsymbol{c_2})) = p_2$*, then the probability distribution* P *with*

$$P(\phi(\boldsymbol{c_1}) \mid \psi(\boldsymbol{c_1})) = P(\phi(\boldsymbol{c_2}) \mid \psi(\boldsymbol{c_2})) = \frac{p_1 + p_2}{2}$$

yields a higher entropy than $\mathsf{SRME}_1(R)$ *but still fulfills Equation (2).* □

Proposition 3. SRME_1 *satisfies (Compatibility I).*

Proof. Let R *be a ground knowledge base. Now, only Equation (1) is used for determining the space of probability distributions, so* $\models_D^{cp}$ *is equivalent to* $\models$ *in the propositional case, cf. Section 2. Then Equation (3) also becomes equivalent to the propositional case and it is* $\mathsf{ME}(R')(\xi) = \mathsf{SRME}_1(R)(\xi)$ *for any ground sentence* ξ*.* □

Proposition 4. SRME_1 *satisfies (Compatibility II).*

Proof. This is ensured by Equation (1). □

Proposition 5. SRME_1 *satisfies (Convergence).*

Proof. (Sketch) This property follows from (Prototypical Indifference). When the number of constants grows towards infinity, most of the instantiations of a statistical assessment have the same probability in $\mathsf{SRME}_1(R)$ *and in order to fulfill Equation (1) these probabilities must converge to* α*.* □

Proposition 6. SRME_1 *satisfies (Compensation).*

Proof. Let R *be a knowledge base and* $(\phi(\boldsymbol{X}) \mid \psi(\boldsymbol{X}))[\alpha] \in R$ *a statistical assessment with* $\alpha \in (0, 1)$*. Suppose*

$$\mathsf{SRME}_1(R)(\phi(\boldsymbol{c}) \mid \psi(\boldsymbol{c})) < \alpha$$

for all $(\phi(\boldsymbol{c}) \mid \psi(\boldsymbol{c}))[\alpha] \in \mathsf{ground}_D(\phi(\boldsymbol{X}) \mid \psi(\boldsymbol{X}))$*. Then (for finite* D*) it is*

$$\begin{aligned} & \frac{\sum_{(\phi(\boldsymbol{c}) \mid \psi(\boldsymbol{c})) \in \mathsf{ground}_D((\phi(\boldsymbol{X}) \mid \psi(\boldsymbol{X})))} P(\phi(\boldsymbol{c}) \mid \psi(\boldsymbol{c}))}{|\mathsf{ground}_D(\phi(\boldsymbol{X}) \mid \psi(\boldsymbol{X}))|} \\ < & \frac{\alpha \cdot |\mathsf{ground}_D(\phi(\boldsymbol{X}) \mid \psi(\boldsymbol{X}))|}{|\mathsf{ground}_D(\phi(\boldsymbol{X}) \mid \psi(\boldsymbol{X}))|} \\ = & \alpha \end{aligned}$$

contradicting $\mathsf{SRME}_1(R) \models_D^{cp} R$*.* □

5 Discussion and Related Work

The work discussed in this paper is at a preliminary stage and more investigations and experiments have to be undertaken in order to demonstrate usability and usefulness of the proposed approach. To this end the KReate project[1] investigates different approaches for combining relational representations and probabilistic reasoning under maximum entropy. Within this project there are some approaches of applying maximum entropy methods to relational knowledge bases without taking into account statistical information explicitly. Loh [15] and Fisseler [14] both employ the principle of maximum entropy directly on a grounded version of the relational knowledge base. In [15] inconsistencies in the grounded knowledge base are handled by removing contradictory instances of the individual conditionals. This results in a consistent propositional knowledge base for which the probability distribution with maximum entropy can be computed as in the propositional case. Consequently, these approaches treat conditionals with variables as schemas for their instances and thus use only an interpretation of conditionals based on the degree of belief. Nevertheless, it seems that even these approaches satisfy all properties discussed in this paper except (Compensation). So it seems reasonable to assume that the list of properties is incomplete for describing the intuition behind statistical assessments as well as uncertainty assessments. For future work we plan to investigate these properties in more depth and find new properties that characterize this intuition.

6 Summary

In this paper we investigated relational probabilistic reasoning from the point of view of maximum entropy methods and by taking into account the differences of statistical information and degrees of belief. We defined common sense properties for inference in first-order probabilistic conditional logic that represent this distinction and proposed an inference operator that fulfills this properties. Finally, we closed with some discussions.

As mentioned above the work reported here is at a preliminary stage and further investigations of the topic are mandatory. A comprehensive comparison of our approach and the approaches discussed in the previous section is part of current research.

Acknowledgements. The author thanks the reviewers for their helpful comments to improve the original version of this paper. Special thanks go to Gabriele Kern-Isberner for discussions and helpful suggestions. The research reported here was partially supported by the Deutsche Forschungsgemeinschaft (grant KE 1413/2-1).

[1] `http://www.fernuni-hagen.de/wbs/research/kreate/index.html`

References

1. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1998)
2. De Raedt, L., Kersting, K.: Probabilistic inductive logic programming. In Raedt, L.D., Kersting, K., Landwehr, N., Muggleton, S., Chen, J., eds.: Probabilistic Inductive Logic Programming. Volume 4911 of Lecture Notes in Computer Science. Springer (2008) 1–27
3. Cussens, J.: Logic-based Formalisms for Statistical Relational Learning. In Getoor, L., Taskar, B., eds.: Introduction to Statistical Relational Learning. MIT Press, Cambridge, MA (2007)
4. Kersting, K., De Raedt, L.: Bayesian Logic Programming: Theory and Tool. In Getoor, L., Taskar, B., eds.: An Introduction to Statistical Relational Learning. MIT Press (2005)
5. Domingos, P., Richardson, M.: Markov Logic: A Unifying Framework for Statistical Relational Learning. In: Proceedings of the ICML-2004 Workshop on Statistical Relational Learning and its Connections to Other Fields, Banff, Canada (2004) 49–54
6. Wellman, M.P., Breese, J.S., Goldman, R.P.: From Knowledge Bases to Decision Models. The Knowledge Engineering Review **7**(1) (1992) 35–53
7. Breese, J.S.: Construction of Belief and Decision Networks. Computational Intelligence **8**(4) (1992) 624–647
8. Nute, D., Cross, C.: Conditional logic. In Gabbay, D., Guenther, F., eds.: Handbook of Philosophical Logic. Volume 4. Kluwer (2002) 1–98
9. Benferhat, S., Dubois, D., Prade, H.: Possibilistic and Standard Probabilistic Semantics of Conditional Knowledge Bases. Journal of Logic and Computation **9**(6) (1999) 873–895
10. Grove, A.J., Halpern, J.Y., Koller, D.: Random Worlds and Maximum Entropy. Journal of Artificial Intelligence Research (JAIR) **2** (1994) 33–88
11. Paris, J.: The Uncertain Reasoner's Companion: A Mathematical Perspective. Cambridge University Press (1994)
12. Kern-Isberner, G.: Conditionals in Nonmonotonic Reasoning and Belief Revision. Number 2087 in Lecture Notes in Computer Science. Springer (2001)
13. Lukasiewicz, T., Kern-Isberner, G.: Probabilistic Logic Programming under Maximum Entropy. In: Proceedings ECSQARU-99. Volume 1638., Springer Lecture Notes in Artificial Intelligence (1999) 279–292
14. Fisseler, J.: Learning and Modeling with Probabilistic Conditional Logic. PhD thesis, FernUniversität in Hagen, Germany (to appear 2009)
15. Loh, S.: Relational Probabilistic Inference Based on Maximum Entropy. Master's thesis, Technische Universität Dortmund (to appear 2009)
16. Halpern, J.Y.: An Analysis of First-Order Logics of Probability. Artificial Intelligence **46** (1990) 311–350
17. Bacchus, F., Grove, A.J., Halpern, J.Y., Koller, D.: From Statistical Knowledge Bases to Degrees of Belief. Artificial Intelligence **87**(1–2) (1996) 75–143
18. Friedman, N., Halpern, J.Y., Koller, D.: First-order conditional logic revisited. In: AAAI/IAAI, Vol. 2. (1996) 1305–1312
19. Delgrande, J.P.: On First-Order Conditional Logics. Artificial Intelligence **105**(1–2) (1998) 105–137
20. Rödder, W., Meyer, C.H.: Coherent Knowledge Processing at Maximum Entropy by SPIRIT. In: Proc. of the Twelfth Conf. on Uncertainty in Artificial Intelligence. (1996) 470–476

21. Fisseler, J.: Toward Markov Logic with Conditional Probabilities. In Wilson, D.C., Lane, H.C., eds.: Proceedings of the Twenty-First International FLAIRS Conference, AAAI Press (2008) 643–648
22. Beachy, J.A., Blair, W.D.: Abstract Algebra. third edn. Waveland Press, Inc., Long Grove, Illinois, USA (2005)
23. Bernoulli, J.: Usum & Applicationem Praecedentis Doctrinae in Civilibus, Moralibus & Oeconomicis. In: Ars Conjectandi. (1713)

Reinforcement Learning for Golog Programs

Daniel Beck and Gerhard Lakemeyer

Department of Computer Science
RWTH Aachen University
Germany
{dbeck|gerhard}@cs.rwth-aachen.de

Abstract. A special feature of programs in the action language Golog are non-deterministic actions, which require an agent to make choices during program execution. In the presence of stochastic actions and rewards, Finzi and Lukasiewicz have shown how to arrive at optimal choices using reinforcement learning techniques applied to the first-order MDP representations induced by the program. In this paper we extend their ideas in two ways: we adopt a first-order SMDP representation, which allows Q-updates to be limited to the non-deterministic choice points within a program, and we give a completely declarative specification of a learning Golog interpreter.

1 Introduction

In classical reinforcement learning (RL) and Markov Decision Processes (MDPs), we are given a set of states, actions which stochastically take us from a given state into one of a number of states, and a reward function over states. The goal of learning is to find the optimal policy, which tells us for each state which action to select to maximize our expected reward. In principle this is well understood with methods such as Q-learning solving the problem. However, for most practical applications the huge state and action space is a concern, as explicit representations usually are not viable computationally. To address this problem, state abstraction mechanisms have been explored [1], including FOMDPs [2], which employ first-order logic to characterize a possibly infinite state space using a finite set of formulas.

In this paper, we take this idea further by also constraining the action space using programs written in the action language Golog. Roughly, instead of a state and a set of primitive actions to choose from, we are given a formula describing the current state and a program we need to follow. In the extreme case, when the program is completely deterministic, there is nothing to learn, as the program tells us exactly what the next action is. However, in general the program allows for non-deterministic choices, and here we again need to learn what choices are the best ones in terms of maximizing expected rewards. As we will see, the idea of Q-learning can be adapted to this setting.

More precisely, based on earlier work [2, 3], we start by presenting a method to compute, for a given reward function and Golog program, first-order state

formulas describing the possible states before the program is executed. Roughly, these formulas specify sets of states which are equivalent in the sense that the expected rewards are identical when following a policy which is compliant with the program. Moreover, only those properties of the states which are relevant to the expected reward are reflected in those state formulas.

In a way similar to [1], we then construct a joint semi-MDP (SMDP) over a state space which is made up of tuples consisting of a subprogram of the given program which starts off with a non-deterministic choice and a corresponding state formula.

Lastly, we give the semantics for our new Golog dialect QGolog which incorporates reinforcement learning techniques to learn the optimal decisions for the choice points of a program by means of executing it and observing the outcomes. In essence, we integrate a Q-learning algorithm for the SMDP described above. While [3] also considers a form of Q-learning, their approach is different in that they ignore the SMDP-nature of Golog programs. Perhaps more importantly, we give a completely declarative specification of learning, which we feel is more transparent and better lends itself to formal analysis.

We remark that, since the semantics of Golog requires to axiomatize the dynamics of actions, it is not possible to be completely model-free as in standard RL. Thus, we still assume that the effects of deterministic actions (and thus the successor states) are known; also, the possible outcomes of a stochastic action are known. But we do not assume that the probability distribution over these outcomes is known.

The rest of the paper is organized as follows. After giving a very brief introduction to the situation calculus and Golog, Sect. 3 considers the generation of state-partition formulas. In Sect. 4, we discuss the SMDP induced by a Golog program and specify how Q-learning works in this setting. We then present first experimental results, discuss related work and conclude.

2 Foundations

2.1 The Situation Calculus and Golog

The *situation calculus* is a sorted first-order[1] language with equality and sorts of type action and situation. A situation is a history of executed actions; the initial situation is denoted by S_0; the successor situation which results from executing action a in situation s is denoted as $do(a, s)$. Properties of the world that might change from situation to situation are described by means of (relational) *fluents,* which are ordinary predicate symbols which have a situation term as their last argument. Formulas which mention only a single situation term σ and which do not quantify over situations are called *uniform* in σ. We sometimes consider *situation-suppressed* formulas which are obtained by removing all situation arguments from the fluents. If ϕ is a situation-suppressed formula, then $\phi[\sigma]$ denotes the formula which restores the situation σ in all the fluents mentioned.

[1] There are also some second-order features, which do not concern us here.

The preconditions for each action A are given by $Poss(A(\boldsymbol{x}), s) \equiv \Pi_A(\boldsymbol{x}, s)$.[2] According to Reiter's solution of the frame problem [4] the effects of actions are encoded as so-called *successor-state axioms (SSAs),*one for each fluent:

$$F(\boldsymbol{x}, do(a, s)) \equiv \Phi_F(\boldsymbol{x}, a, s).$$

A *basic action theory (BAT)* $\mathcal{D}$ consists of the foundational axioms Σ, which define the space of situations, the successor state axioms $\mathcal{D}_{ssa}$, the action preconditions D_{ap}, the unique name axioms for actions $\mathcal{D}_{una}$, and a set $\mathcal{D}_{S_0}$ of first-order sentences uniform in S_0 which describe the fluent values in the initial situation.

Example. Consider the blocks world domain. The fluent $on(b_1, b_2, s)$ expresses that block b_1 is on top of block b_2. The action $move(b_1, b_2)$ moves block b_1 on block b_2. It can only be performed iff there is, in the current situation, no other block on b_1 and on b_2 except if b_2 is the table. Furthermore, b_1 and b_2 have to be distinct.

$$Poss(move(b_1, b_2), s) \equiv \neg\exists z.on(z, b_1, s) \wedge \\ (b_2 \neq table \supset \neg\exists z.on(z, b_2, s)) \wedge b_1 \neq b_2$$

A block b_1 is on top of b_2 iff it has just been moved there or iff it has been there before and wasn't moved away with the last action.

$$on(b_1, b_2, do(a, s)) \equiv a = move(b_1, b_2) \vee on(b_1, b_2, s) \wedge \neg\exists z.a = move(b_1, z)$$

Besides deterministic primitive actions like *move* we also include stochastic actions. The idea is that, when a stochastic action is executed, nature chooses one of a finite number of deterministic actions [5]. Formally, for a stochastic action a_s the possible choices of primitive actions $n_1, \ldots, n_k$ are defined as

$$choice(a_s(\boldsymbol{x}), a) \equiv \bigvee_{i=1}^{k} a = n_i(\boldsymbol{x}).$$

We denote the probability with which $n_i(\boldsymbol{x})$ is chosen as the outcome of action $a_s(\boldsymbol{x})$ in situation s by $prob(n_i(\boldsymbol{x}), a_s(\boldsymbol{x}), s)$. Axioms of the form

$$\sum_{i=1}^{k} prob(n_i(\boldsymbol{x}), a_s(\boldsymbol{x}), s) = 1$$

ensure that we indeed obtain proper probability distributions.

If the probability distribution with which nature chooses is known, this can also be specified. In our setting, the distribution is generally not known. Moreover, the distributions may change from situation to situation, but not arbitrarily. We assume that there are situation suppressed formulas $\theta_1, \ldots, \theta_r$, which

[2] In formulas like these free variables are understood to be implicitly universally quantified.

partition the set of situations (see Definition 1 below for what that means) so that situations which satisfy the same θ_j agree on the distribution over the n_i. Formally, we include axioms of the form

$$\theta_j(s) \wedge \theta_j(s') \supset prob(n_i(\boldsymbol{x}), a_s(\boldsymbol{x}), s) = prob(n_i(\boldsymbol{x}), a_s(\boldsymbol{x}), s').$$

Note that in the simple case where the distribution does not change at all, there is only one $\theta = true$.

To ensure full observability it has to be possible to determine the actual outcome of a stochastic action. Therefore, sensing conditions $senseCond(n_i) \equiv \varphi_i$ are defined such that by means of the special action $senseEffect(a_s)$ the truth value of φ_i and thus the actual outcome can be determined.

The regression of a formula ϕ through an action a is a formula ϕ'. The idea is that, for a given BAT, ϕ holds after executing a just in case ϕ' held before the execution of a. Suppose that the SSA for fluent F is $F(\boldsymbol{x}, a, s) \equiv \Phi_F(\boldsymbol{x}, a, s)$. Then we inductively define the regression of a formula which is uniform in the situation $do(a, s)$ as:

$$\begin{aligned}
&Regr(F(\boldsymbol{x}, do(a, s)) = \Phi_F(\boldsymbol{x}, a, s)\\
&Regr(\neg\phi) = \neg Regr(\phi)\\
&Regr(\phi_1 \wedge \phi_2) = Regr(\phi_1) \wedge Regr(\phi_2)\\
&Regr(\exists x.\phi) = \exists x.Regr(\phi)
\end{aligned}$$

According to the regression theorem (Theorem 4.5.4 in [5]) two formulas $\phi(\boldsymbol{x}, s)$ and $\phi'(\boldsymbol{x}, do(a, s))$ where $\phi(\boldsymbol{x}, s) = Regr(\phi'(\boldsymbol{x}, do(a, s)))$ are logically equivalent wrt a given BAT, that is, $\mathcal{D} \models \phi \equiv \phi'$.

The high-level agent programming language Golog [6] is based on the situation calculus. Roughly, Golog allows us to write programs where the primitive actions are those defined by a basic action theory. Here we consider the following language constructs: primitive actions (a), test actions ($\varphi?$), sequences ($[\delta_1; \delta_2]$), conditionals (**if** φ **then** δ_1 **else** δ_2 **end**), loops (**while** φ **do** δ **end**), non-deterministic choice ($[\delta_1 \,|\, \delta_2]$), non-deterministic choice of arguments (**pick**$(x, \delta(x))$), non-deterministic iteration (δ^*), and procedures (**proc** $P(\boldsymbol{x})$ δ **end**).

The meaning of a Golog program can be defined with the help of two special predicates $Final(\delta, s)$ and $Trans(\delta, s, \delta', s')$, which can be read as "$\delta$ can legally terminate in situation s" and "executing the first action of program δ in situation s leads to situation s' with remaining program δ'." For example, if A is a primitive action, then $Trans([A; \rho], s, \delta', s')$ holds iff $Poss(A, s)$ holds, $s' = do(A, s)$, and $\delta' = \rho$. For lack of space, we will define $Trans$ only for the new constructs introduced in this paper and refer to [7] for the others. To start with, for a stochastic action a_s, $Trans$ is defined as $Trans(a_s, s, \delta, s') \equiv s = s' \wedge \delta = senseEffect(a_s)$, that is, a_s is simply replaced by the sensing action, which senses the actual outcome of the action. The idea is that, when $senseEffect(a_s)$ is executed by the Golog interpreter, a_s gets executed first before the sensing takes place. See the description of the interpreter (Section 4) for how this is done.

2.2 Markov Decision Processes

A *Markov Decision Process* (MDP) is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ a set of actions, $\mathcal{T}(s, a, s')$ assigns probabilities to the transitions from state s to state s' with action a, and $\mathcal{R}(s, a, s')$ assigns a reward to getting from s to s' by performing action a. A solution of a MDP is represented by a policy π which maps states to actions; π^* is the optimal policy and achieves the maximal expected reward. A semi-MDP (SMDP) allows the actions to have different durations. Then, $\mathcal{T}$ defines a mapping $\mathcal{S} \times N \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ where N are the natural numbers. It specifies the probability of getting from a state s in n time steps to state s' by performing action a.

3 Generating Partitions

For a set of situation suppressed state formulas $S(\boldsymbol{x}) = \{\phi_1(\boldsymbol{x}), \ldots, \phi_n(\boldsymbol{x})\}$, $S(\boldsymbol{x})[s]$ denotes the set of state formulas $S' = \{\phi_i(\boldsymbol{x})[s] \,|\, 1 \leq i \leq n\}$.

In the following we make the assumption that $\mathcal{D}_{S_0}$ is *completely specified*, i.e., for every state formula ϕ either $\mathcal{D}_{S_0} \models \phi[S_0]$ or $\mathcal{D}_{S_0} \models \neg\phi[S_0]$ holds.

Definition 1. *A set $P(\boldsymbol{x}) = \{\phi_1(\boldsymbol{x}), \ldots, \phi_n(\boldsymbol{x})\}$ of state formulas ϕ_i is a partition iff the following conditions hold for all completely specified $\mathcal{D}_{S_0}$ and an arbitrary ground situation term σ:*

1. $\mathcal{D} \models \forall \boldsymbol{x}. \bigvee_{i=1}^{n} \phi_i(\boldsymbol{x})[\sigma]$ *and*
2. $\mathcal{D} \models \forall \boldsymbol{x}. \phi_i(\boldsymbol{x})[\sigma] \supset \neg \left(\bigvee_{j \neq i} \phi_j(\boldsymbol{x})[\sigma] \right)$.

Definition 2. *Let $S_1(\boldsymbol{x}_1)$ and $S_2(\boldsymbol{x}_2)$ be sets of state formulas. Then, $S_1(\boldsymbol{x}_1) \otimes S_2(\boldsymbol{x}_2)$ is defined as:*

$$S_1(\boldsymbol{x}_1) \otimes S_2(\boldsymbol{x}_2) = \{\phi^1(\boldsymbol{x}_1) \wedge \phi^2(\boldsymbol{x}_2) \,|\, \phi^1 \in S_1, \phi^2 \in S_2\}$$

Lemma 1. *If $P_1(\boldsymbol{x}_1)$ and $P_2(\boldsymbol{x}_2)$ are state partitions then $P_1(\boldsymbol{x}_1) \otimes P_2(\boldsymbol{x}_2)$ is also a partition according to Def. 1. Also, $\{(\{\phi\} \otimes P_1), \neg\phi\}$ and $\{(\{\phi\} \otimes P_1), (\{\neg\phi\} \otimes P_2)\}$ are partitions.*

3.1 The Reward Partition

The reward function $rew(s)$ defines a mapping from the space of situations into the reals. We assume that the reward function can be written in the following form:

$$rew(s) = r \equiv \phi_1^{rew}[s] \wedge r = r_1 \vee \ldots \vee \phi_m^{rew}[s] \wedge r = r_m$$

where the r_i are distinct numeric constants. To ensure that $rew(s)$ is well-defined it is necessary to require that $P^{rew} = \{\phi_1^{rew}, \ldots, \phi_m^{rew}\}$ is a partition.

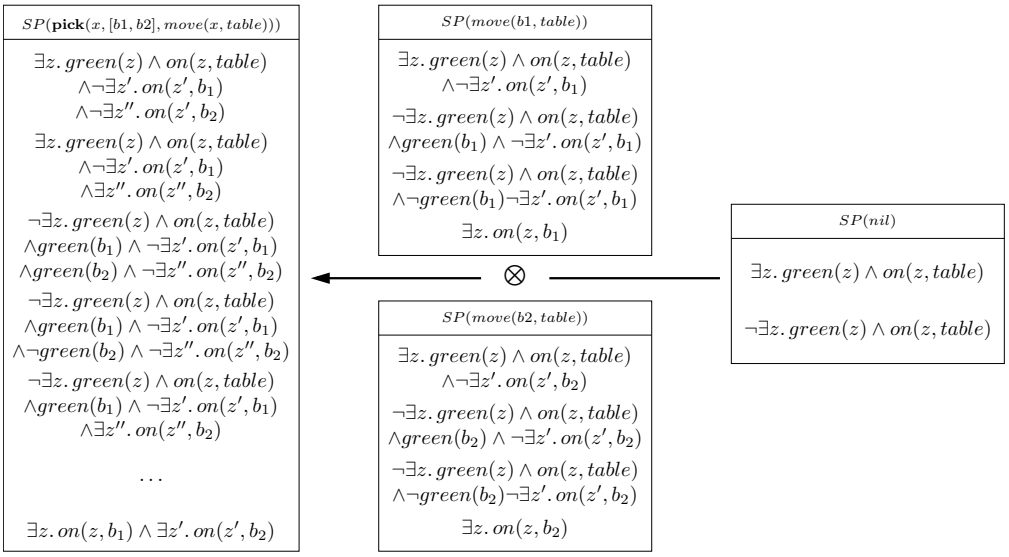

Fig. 1. The partition induced by the program $\mathbf{pick}(x,[b1,b2],move(x,table))$. In case there is a green block on the table a reward of 10 is received otherwise the reward is 0. Consequently, the reward partition is $P^{rew} = \{\exists z.green(z) \wedge on(z,table), \neg\exists z.green(z) \wedge on(z,table)\}$.

3.2 Partitions Induced by Golog Programs

Given a reward function $rew(s)$ a Golog program δ induces a partition which separates those states from each other that (potentially) have different expected rewards when executing program δ. The partition that is induced by the program δ given the reward function $rew(s)$ is denoted by $SP(\delta \,|\, rew(s))$ (often we omit to explicitly mention the reward function when it is clear what reward function is meant and just write $SP(\delta)$).

The partition $SP(\delta)$ is recursively defined over the structure of the remaining program. Consequently, $SP(\delta)$ is not well-defined for programs that have infinite execution traces. To avoid those we preprocess the programs and replace potentially dangerous constructs. In particular, these are:

- $[\mathbf{while}\ \varphi\ \mathbf{do}\ \delta'\ \mathbf{end}; \delta]$ is replaced by a finite number of nested conditionals:

$$[\mathbf{if}\ \varphi\ \mathbf{then}\ [\delta'; \mathbf{if}\ \varphi\ \mathbf{then}\ [\delta'; \ldots] \\ \mathbf{else}\ nil\ \mathbf{end}]\ \mathbf{else}\ nil\ \mathbf{end}; \delta]$$

- The star-operator is reduced to a non-deterministic choice between a finite number of repetitions: δ^* is replaced by $\mathbf{nondet}(nil, \delta, \delta^2, \ldots, \delta^n)$, where δ^i stands for the i-fold repetition of δ.

Further, we have to disallow recursive procedure calls in order to guarantee a finite execution trace. Moreover, we assume that the program is nil-terminated, i.e., it has the form $[\gamma; nil]$ where γ is an arbitrary (but finite) Golog program.

The partitions induced by this class of finite Golog programs can then be defined as follows.

The partition induced by the **empty program** is given by the reward partition, i.e., the (empty) program does not affect the expected reward.

$$SP(nil) = P^{rew}.$$

For programs that start with a **sequence** the first element of the sequence determines the induced partition.

$$SP([[\delta_1; \delta_2]; \delta_3]) = SP([\delta_1; [\delta_2; \delta_3]])$$

To determine the partition induced by a program that starts with a **primitive action** a regression is the key. Regression allows to compile state formulas which describe the state before executing a from the state formulas in the partition induced by the remaining program. Further, we make sure that the action's preconditions hold, split up the state formulas according to the reward partition, and complete the partition by adding a state formula for the case where the preconditions are not given.

$$SP([a(\boldsymbol{x}); \delta])[s] = \{\{Regr(\phi_i[do(a(\boldsymbol{x}), s)]) \wedge Poss(a(\boldsymbol{x}), s) \mid \phi_i \in SP(\delta)\} \otimes P^{rew}[s]\} \cup \{\neg Poss(a(\boldsymbol{x}), s)\}$$

A leading **test action** introduces a further distinguishing feature to the partition induced by the remaining program: either the test condition holds or it does not.

$$SP([\varphi?; \delta]) = \{\varphi\} \otimes SP(\delta) \cup \{\neg\varphi\}$$

Analogously, the partition induced by a program starting with a **conditional** is defined as:

$$SP([\textbf{if } \varphi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ end}; \delta]) = \{\varphi\} \otimes SP([\delta_1; \delta]) \cup \{\neg\varphi\} \otimes SP([\delta_2; \delta])$$

The model of a **stochastic action** is described by a number of possible outcomes and and a situation-dependent probability distribution over those. As already said above the ϑ which define the probability distribution over the outcomes of the stochastic actions a_s form a partition: $P^{pr}_{a_s} = \{\vartheta_1, \ldots, \vartheta_r\}$.

The partition induced by a program starting with a stochastic action can then be defined as:

$$SP([a_s(\boldsymbol{x}); \delta]) = P^{pr}_{a_s} \otimes \left(\bigotimes_{i=1}^{k} SP([n_i(\boldsymbol{x}); \delta]) \right)$$

In case of a **non-deterministic branching** the partition is made up of the combination of the partitions induced by each of the possible branches.

$$SP([[\delta_1 \mid \delta_2]; \delta]) = SP([\delta_1; \delta]) \otimes SP([\delta_2; \delta])$$

Since we restricted the **non-deterministic choice of argument** to a selection from a given set of arguments the partition induced by a program beginning with a non-deterministic choice of argument is the combination of the partitions induced by the remaining program with the different arguments:

$$SP([\mathbf{pick}(x, [v_1, \ldots, v_n], \gamma); \delta]) = \bigotimes_{i=1}^{n} SP([\gamma^x_{v_i}; \delta])$$

For a program starting with a procedure call the partition it induces is computed as the partition induced by a program where the name of the procedure is replaced with its body. Assume a procedure P is defined as **proc** $P(\boldsymbol{x})$ δ_P **end**, then

$$SP([P(\boldsymbol{t}); \delta]) = SP([\delta_P{}^{\boldsymbol{x}}_{\boldsymbol{t}}; \delta]).$$

Theorem 1. *For a Golog program δ and a reward function $rew(s)$, $SP(\delta \mid P^{rew})$ describes a state partition according to Def. 1.*

Proof. by induction on the structure of the program.

1. $SP(nil)$ is a partition by definition.
2. For the proof that $SP([a; \delta])$ is a partition we assume a slightly different definition of $SP([a; \delta])$. The one given below is "finer grained" since it also partitions the cases where $Poss(a, s)$ does not hold.

$$SP(a(\boldsymbol{x}); \delta)[s] = \{Regr(\phi[do(a(\boldsymbol{x}), s)]) \mid \phi \in SP(\delta)\} \\ \otimes \{Poss(a(\boldsymbol{x}), s), \neg Poss(a(\boldsymbol{x}), s)\} \otimes P^{rew}[s]$$

   According to the regression theorem $\{Regr(\phi[do(a(\boldsymbol{x}), s)]) \mid \phi \in SP(\delta)\}$ is a partition iff $SP(\delta)$ is a partition. Then, according to Lemma 1, $SP([a; \delta])$ as given above is also a partition. This also holds for the original, "coarser" definition of $SP([a; \delta])$ since it combines all cases where the preconditions are not given into a single state formula and does not partition this case further as it is done by the alternative definition above.
3. According to Lemma 1 $SP([\varphi?; \delta])$, $SP([\mathbf{if}\ \varphi\ \mathbf{then}\ \delta_1\ \mathbf{else}\ \delta_2\ \mathbf{end}; \delta])$, $SP([a_s(\boldsymbol{x}); \delta])$, and $SP([[\delta_1 \mid \delta_2]; \delta])$ are partitions iff $SP(\delta)$ is a partition.

Theorem 2. *The space abstraction defined by the partitioning scheme is* safe*, i.e., for a program δ, ground situations σ_1 and σ_2 it holds that if $\mathcal{D} \models \phi[\sigma_1]$ and $\mathcal{D} \models \phi[\sigma_2]$ for a $\phi \in SP(\delta)$ then the expected reward for executing δ in σ_1 and in σ_2 is the same.*

An example for the incremental generation of the partition induced by a Golog program and a reward function is depicted in Fig. 1.

4 The QGolog Interpreter

In this section we describe how the joint SMDP alluded to in the introduction is defined exactly, how Q-learning works for such an SMDP, and we show how that is embedded in our QGolog interpreter by specifying its formal semantics.

The statespace $\mathcal{S}$ of the SMDP underlying the Golog program consists of tuples $\langle \phi, \delta \rangle$ where δ is a choicepoint in the program, i.e., a subprogram that begins with either a non-determinstic branching or a non-deterministic choice of argument, and $\phi \in SP(\delta)$. For all $\langle \phi, \delta \rangle \in \mathcal{S}$, $\mathcal{A}(\langle \phi, \delta \rangle)$ is the set of possible actions in the state $\langle \phi, \delta \rangle$. If $\delta = [[\delta_1 \,|\, \delta_2]; \delta']$ then $\mathcal{A}(\langle \phi, \delta \rangle) = \{\delta_1, \delta_2\}$; if $\delta = [\mathbf{pick}(x, [v_1, \ldots, v_n], \gamma); \delta']$ then $\mathcal{A}(\langle \phi, \delta \rangle) = \{\gamma^x_{v_1}, \ldots, \gamma^x_{v_n}\}$. By uniquely identifying the SMDP-actions by Golog programs which correspond to the resepective choices at the choicepoints it is possible to derive a legal execution trace of the program from a given policy for the SMDP. Particularly, the optimal choices at the choicepoints in the program can be derived from the optimal SMDP-policy.

The rewards received for performing a SMDP-action are computed as the sum of the rewards obtained for executing a sequence of primitive actions which leads the program to the next choicepoint in the program and which corresponds to the SMDP-action under consideration. Let k be the number of primitive actions in that sequence then

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{k-1} r_{t+k-1}$$

The update of the Q-value for performing action A_t in state $\langle \phi, \delta \rangle_t$ can then be formulated as

$$Q(\langle \phi, \delta \rangle_t, A_t) \leftarrow Q(\langle \phi, \delta \rangle_t, A_t) + \alpha \cdot \left(R_t + \gamma^k \cdot \max_{A \in \mathcal{A}(\langle \phi, \delta \rangle)} Q(\langle \phi, \delta \rangle_t, A) - Q(\langle \phi, \delta \rangle_t, A_t) \right)$$

where α is the learning rate and k the number of primitive actions executed while performing A_t.

The Q-table storing the Q-values for all state-action pairs is realized as a fluent $q(\phi, \delta, A, v, s)$ that is initialized in D_{S_0} in such a way that all state-action pairs in the SMDP from above are assigned a value. The value for a particular pair can be updated with the action $setQ(\phi, \delta, A, v)$:

$$q(\phi, \delta, p, v, do(a, s)) \equiv a = setQ(\phi, \delta, p, v) \vee q(\phi, \delta, p, v, s) \wedge \neg \exists v'. a = setQ(\phi, \delta, p, v')$$

Likewise, the fluent $\epsilon(s)$ denotes the probability to deviate from the current policy and to explore another action in situation s. The action $setEpsilon(p)$ changes the fluent's value. Moreover, we assume that the sensing action $senseRnd(r)$ returns a random number $r \in [0, 1]$.

The new construct $\mathbf{learn}(\delta)$ initiates the learning for the program δ. In a program configuration $\langle \mathbf{learn}(\delta), s \rangle$ the program may proceed to a configuration

where the situation is unchanged and the remaining program equals the policy computed by the predicate $QDo(\phi^*, \delta^*, A^*, \delta, s, k, r, \pi)$. The arguments of QDo are the SMDP state $\langle \phi^*, \delta^* \rangle$ corresponding to the last seen choicepoint, the action A^* taken at that state, the current program configuration $\langle \delta, s \rangle$, the number k of primitive actions encountered since the last choicepoint, and the cumulative, discounted reward r obtained since then. Formally:

$$Trans(\mathbf{learn}(\delta), s, \delta', s') \equiv \exists \pi.\, QDo(\phi_{start}, \delta, A_{start}, \delta, s, 0, 0, \pi) \wedge \delta' = \pi \wedge s' = s$$

where $\langle \phi_{start}, \delta \rangle$ is a distinguished start state that is added to $\mathcal{S}$; the only possible action in the state is the start action A_{start}.

The policy computed by QDo is a Golog program that describes a valid continuation of the remaining program. Embedded in the policy are $setQ$ actions to update the Q-table, sensing actions to determine the actual outcome of a stochastic action, etc. Also, the policy handles the exploration of the state space, i.e., with a certain probability a non-optimal action (wrt. the current state-action values) is chosen.

The predicate QDo is defined in dependence on the beginning of the remaining program.

The remaining program is the empty program:

$$Do(\phi^*, \delta^*, A^*, nil, s, k, r, \pi) \stackrel{def.}{=} \pi = [setQ(\phi^*, \delta^*, A^*, r); setEpsilon(\eta \cdot \epsilon)]$$

where $\eta \in (0, 1]$.

The remaining program starts with a primitive action:

$$\begin{aligned} QDo(\phi^*, \delta^*, A^*, [a; \delta], s, k, r, \pi) \stackrel{def.}{=}\ & Poss(a, s) \wedge \exists r_t, \pi'.\, r_t = rew(s) \\ & \wedge QDo(\phi^*, \delta^*, A^*, \delta, do(a, s), k + 1, r + \gamma^k \cdot r_t, \pi') \\ & \wedge \pi = [a; \pi'] \\ & \vee \neg Poss(a, s) \wedge \pi = setQ(\phi^*, \delta^*, A^*, r_{fail}) \end{aligned}$$

r_{fail} is a domain independent negative reward to punish the unsuccessful execution of the program.

The remaining program starts with a test action:

$$\begin{aligned} QDo(\phi^*, \delta^*, A^*, [\varphi?; \delta], s, k, r, \pi) \stackrel{def.}{=}\ & \varphi[s] \wedge QDo(\phi^*, \delta^*, A^*, \delta, s, k, r, \pi) \\ & \vee \neg\varphi[s] \wedge \pi = setQ(\phi^*, \delta^*, A^*, r_{fail}) \end{aligned}$$

The remaining program starts with a stochastic action:

$$\begin{aligned} & QDo(\phi^*, \delta^*, A^*, [a_s; \delta], s, k, r, \pi) \stackrel{def.}{=} \\ & \quad \pi = [a_s; senseEffect(a_s); \\ & \qquad \textbf{if } \varphi_1 \textbf{ then} \\ & \qquad\quad \mathbf{l}(\phi^*, \delta^*, A^*, \delta, k + 1, r + \gamma^k \cdot rew(do(n_1, s)) \\ & \qquad \textbf{elsif } \varphi_2 \textbf{ then } \ldots \\ & \qquad \textbf{end}] \end{aligned}$$

where the φ_i are the sensing conditions for the outcomes n_i of the stochastic action a_s. Here, the computation of the policy is interrupted since it is necessary to determine the actual outcome of executing a_s first, before the policy for the remaining program δ is computed. The construct $\mathbf{l}(\ldots)$ mentioned by the policy is comparable to **learn**$(\ldots)$ but it allows to memorize the last SMDP state, the action taken there, and the number of primitive actions performed since then and the reward obtained for those.

$$Trans(\mathbf{l}(\phi^*, \delta^*, A^*, \delta, k, r), s, \delta', s') \equiv \exists \pi.\, QDo(\phi^*, \delta^*, A^*, \delta, s, k, r, \pi) \\ \wedge \delta' = \pi \wedge s' = s$$

This way the computation of the policy can be continued after executing the stochastic action and observing its outcome.

The remaining program starts with a conditional:

$$\begin{aligned} &QDo(\phi^*, \delta^*, A^*, [\textbf{if } \varphi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ end}; \delta], s, k, r, \pi) \stackrel{def.}{=} \\ &\quad \varphi[s] \wedge QDo(\phi^*, \delta^*, A^*, [\delta_1; \delta], s, k, r, \pi) \\ &\quad \vee \neg\varphi[s] \wedge QDo(\phi^*, \delta^*, A^*, [\delta_2; \delta], s, k, r, \pi) \end{aligned}$$

The remaining program starts with a procedure call $P(\boldsymbol{t})$ and the procedure is defined as **proc** $P(\boldsymbol{v})$ δ_P **end**:

$$QDo(\phi^*, \delta^*, A^*, [P(\boldsymbol{t}); \delta], s, k, r, \pi) \stackrel{def.}{=} QDo(\phi^*, \delta^*, A^*, [\delta_P{}^{\boldsymbol{v}}_{\boldsymbol{t}}; \delta], s, k, r, \pi)$$

The remaining program starts with a non-deterministic branching:

$$\begin{aligned} &QDo(\phi^*, \delta^*, A^*, [[\delta_1 \mid \delta_2]; \delta], s, k, r, \pi) \stackrel{def.}{=} \\ &\quad \bigvee_{\phi \in SP([[\delta_1 \mid \delta_2); \delta]} \phi[s] \wedge \exists q_t.\, q(\phi^*, \delta^*, A^*, q_t, s) \\ &\quad \wedge \exists q, A.\, QMax(\phi, [[\delta_1 \mid \delta_2]; \delta], A, q, s) \\ &\quad \wedge \exists q_{t+1}.\, q_{t+1} = q_t + \alpha \cdot \big(r + \gamma^k \cdot q - q_t\big) \\ &\quad \wedge \pi = [setQ(\phi^*, \delta^*, A^*, q_{t+1}); senseRnd(r); \\ &\qquad \textbf{if } r > \epsilon \textbf{ then} \\ &\qquad\quad \mathbf{l}(\phi, [[\delta_1 \mid \delta_2]; \delta], A, [A; \delta], 0, 0) \\ &\qquad \textbf{elsif } r \leq \frac{\epsilon}{2} \textbf{ then} \\ &\qquad\quad \mathbf{l}(\phi, [[\delta_1 \mid \delta_2]; \delta], \delta_1, [\delta_1; \delta], 0, 0) \\ &\qquad \textbf{else} \\ &\qquad\quad \mathbf{l}(\phi, [[\delta_1 \mid \delta_2]; \delta], \delta_2, [\delta_2; \delta], 0, 0) \\ &\qquad \textbf{end}] \end{aligned}$$

Again, the computation of the policy needs to be interrupted here since it needs to be decided randomly whether to explore or to exploit (after updating the Q-value of the last SMDP-state). The auxiliary predicate $QMax(\phi, \delta, A, q_{max}, s)$

determines the action A with the maximal Q-value q_{max} for the state $\langle \phi, \delta \rangle$ in situation s. It is defined as:

$$\begin{aligned} &QMax(\phi, \delta, A, q_{max}, s) \stackrel{def.}{=} \\ &\quad \bigvee_{A' \in \mathcal{A}(\langle \phi, \delta \rangle)} \exists q_{A'}.q(\phi, \delta, A', q_{A'}, s) \wedge \\ &\quad \bigwedge_{\substack{B \in \mathcal{A}(\langle \phi, \delta \rangle), \\ B \neq A'}} \exists q_B.q(\phi, \delta, B, q_B, s) \wedge q_B \leq q_{A'} \wedge \\ &\quad\quad A = A' \wedge q_{max} = q_{A'} \end{aligned}$$

The remaining program starts with a non-deterministic choice of argument:

$$\begin{aligned} &QDo(\phi^*, \delta^*, A^*, [\mathbf{pick}(x, [v_1, \ldots, v_n], \gamma); \delta], s, k, r, \pi) \stackrel{def.}{=} \\ &\quad \bigvee_{\phi \in SP([\mathbf{pick}(x, [v_1, \ldots, v_n], \gamma); \delta])} \phi[s] \wedge \exists q_t.\, q(\phi^*, \delta^*, A^*, q_t, s) \\ &\quad \wedge \exists A, q.\, QMax(\phi, [\mathbf{pick}(x, [v_1, \ldots, v_n], \gamma); \delta], A, q, s) \\ &\quad \wedge \exists q_{t+1}.\, q_{t+q} = q_t + \alpha \cdot \left(r + \gamma^k \cdot q - q_t\right) \\ &\quad \wedge \pi = [setQ(\phi^*, \delta^*, A^*, q_{t+1}); senseRnd(r); \\ &\quad\quad \mathbf{if}\ r > \epsilon\ \mathbf{then} \\ &\quad\quad\quad \mathbf{l}(\phi, [\mathbf{pick}(x, [v_1, \ldots, v_n], \gamma); \delta], A, [A; \delta], 0, 0) \\ &\quad\quad \mathbf{elsif}\ r \leq \frac{1}{n} \cdot \epsilon\ \mathbf{then} \\ &\quad\quad\quad \mathbf{l}(\phi, [\mathbf{pick}(x, [v_1, \ldots, v_n], \gamma); \delta], \gamma^x_{v_1}, [\gamma^x_{v_1}; \delta], 0, 0) \\ &\quad\quad \mathbf{elsif}\ \ldots \\ &\quad\quad \mathbf{elsif}\ \frac{n-1}{n} \cdot \epsilon < r \leq \epsilon\ \mathbf{then} \\ &\quad\quad\quad \mathbf{l}(\phi, [\mathbf{pick}(x, [v_1, \ldots, v_n], \gamma); \delta], \gamma^x_{v_n}, [\gamma^x_{v_n}; \delta], 0, 0) \\ &\quad\quad \mathbf{end}] \end{aligned}$$

5 Evaluation

We implemented the QGolog interpreter in Prolog and tested it in the blocks world domain with the following program which either does nothing (the primitive action *noop*) or non-deterministically picks a block, moves that block on the table, and then moves block b_1 on block b_2.

$$[[\mathbf{pick}(x, [b_2, \ldots, b_5], move(x, table)); move(b1, b2)] | noop]$$

The reward function assigns the value ten to situations where block b_1 is on top of block b_2 and zero to all others. If the program reaches a configuration in which it cannot be executed further since a precondition is not fulfilled a

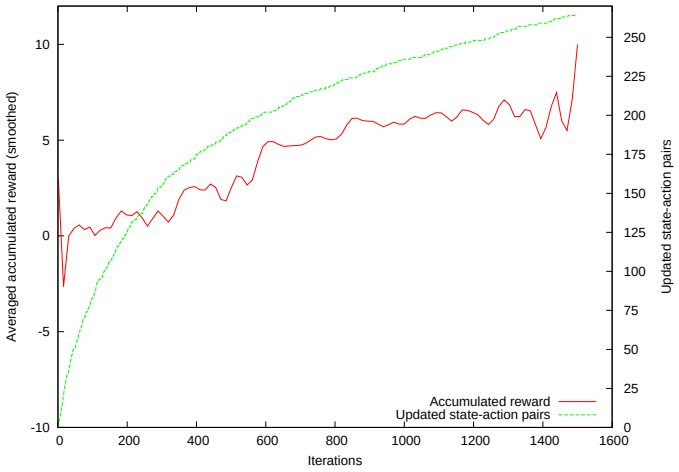

Fig. 2. The graph shows the accumulated reward obtained when executing the program in random, 5-block instances of the blocksworld averaged over 10 runs. For better readability the results are additionally smoothed. Additionally, the number of updated entries in the Q-table is depicted.

negative reward of -10 is given. The values in the Q-table are initialized with -1. Throughout the experiments the learning rate and the exploration probability were kept constant at 0.1.

For each iteration, that is, for each run of the program a new 5-block instance of the blocksworld domain was generated. The results of the experiments are averaged over ten runs and are shown in Fig. 2. For instances in which the program cannot achieve a situation in which block b_1 is on top of block b_2, i.e., the maximally achievable reward is 0 we set the reward to 10 in case an accumulated reward of 0 was achieved to honor the successful execution of the program. Additionally Fig. 2 shows the number of seen state-action combinations during the execution of the program over the course of the experiments.

Although even after the 1500 iterations new state-action combinations are encountered the average accumulated reward crosses the 50% mark after 600 iterations. That is, after 600 iterations the interpreter selects an execution trace that yields an accumulated reward of 5 and above (in average). Admittedly, the results do not look that impressive at first glance. But just reconsider that we have two choicepoints in the program with two and four choices, respectively, and the number of ground configurations of a 5-blocks blocksworld is already in the thousands. That implies that it would take much longer for "flat" Q-learning to explore the state space and find the "good" actions.

Furthermore, the experiments showed that the current way of generating the partition for a **pick** by basically grounding the state formula results in an exponential number of elements in the partitions (wrt the number of elments in the partition induced by the remaining program). Finding a way to characterize the choice of a domain element by a first-order formula would greatly improve the abstraction.

6 Related Work

Restricting the space of policies by means of (partial) programs has been proposed and implemented multiple times. The differences can be mainly found in the expressiveness of the proposed languages in which those partial program are formulated. The HAM-language [8] allows the programmer to define a hierarchical structure of machines whose states can either be action-states which trigger the execution of an action, be a choice state in which the next machine state is selected non-deterministically, or a call state which executes another machine. In [9] the language was extended by parametrization, aborts/interrupts, and memory variables. This raises the expressiveness which allows for more compact programs. These languages were superseded by the language ALisp which extends standard Lisp. In comparison to those languages Golog has a clearly defined semantics which allows to automatically generate abstract state descriptions as it was shown in this paper. State abstraction in ALisp requires the programmer to manually provide abstraction functions [1].

Logic-based representation languages are employed by several approaches for relational reinforcement learning to describe state and action in an abstract fashion. Though, the expressiveness is usually less than the expressiveness of full first-order languages (e.g., quantification is only incorporated implicitly). The approach for symbolic dynamic programming as it was proposed in [2] employs the full expressiveness of a first-order language but at the cost that full theorem proving is required to develop a first-order representation of the value functions. Though, in our approach we make use of the full first-order expressiveness, too, syntactic manipulation of the formulas is sufficient since the structure of the value functions is assumed to be given by the program. This might result in a separation of states which would be joined in the symbolic dynamic programming approach but this is only possible if the complete model is known.

The work which inspired our approach is described in [3]. We refine their approach in several ways. First, we do not employ a horizon in the generation of the partition induced by Golog programs. Only where it is necessary we rewrite the programs to ensures finite execution traces. A consequence thereof is that the horizon is not part of the state description. Secondly, we tightly integrate the reinforcement learning process in the language Golog and do not handle the learning externally. And lastly, we do the Q-update only for the choicepoints and not for every single primitive action. This seems to be reasonable since only at the choicepoints a decsision has to be made—if the program tells the interpreter to execute a primitive action it has no choice. This leads to a faster convergence of the Q-values.

7 Conclusion

In this paper we showed how reinforcment learning can be integrated into the Golog action language framework. We demonstrated how a Golog progam together with its underlying basic action theory gives rise to a first-order SMDP

and we gave a completely declarative specification of a learning Golog interpreter.

In ongoing work we are investigating a number of extensions such as these:

- We want to generalize **pick** to allow for an arbitrary choice of arguments instead of choosing from a finite list of given objects. The advantage would be that state-partition formulas can be generated which are independent of the actual objects in the domain. For example, it would not matter whether the blocks world contained 5 or 500 blocks.
- Perhaps more importantly, we are working on a form of hierarchical reinforcement learning along the lines of [10]. The idea is that procedure calls within programs form a natural hierarchy, and under certain conditions the choices within procedures can be learned independently from those in other procedures. For example, the task of building a tower of a certain color can be divided into collecting blocks of the specified color and then calling a subroutine to build the tower, where learning how to build a tower is completely independent of the color in question.

References

1. Andre, D., Russell, S.J.: State abstraction for programmable reinforcement learning agents. In: AAAI/IAAI. (2002) 119–125
2. Boutilier, C., Reiter, R., Price, B.: Symbolic dynamic programming for first-order MDPs. In: Proceedings of the Seventeenth International. Joint Conference on Artificial Intelligence, IJCAI 2001. (August 2001) 690–700
3. Finzi, A., Lukasiewicz, T.: Adaptive multi-agent programming in GTGolog. In: KI 2006: Advances in Artificial Intelligence, 29th Annual German Conference on AI. Volume 4314., Springer (June 2007) 389–403
4. Reiter, R.: The frame problem in situation the calculus: a simple solution (sometimes) and a completeness result for goal regression. (1991) 359–380
5. Reiter, R.: Knowledge in Action. MIT Press (2001)
6. Levesque, H., Reiter, R., Lesperance, Y., Lin, F., Scherl, R.: Golog: A logic programming language for dynamic domains. The Journal of Logic Programming **31**(1-3) (1997) 59–83 Reasoning about Action and Change.
7. Giacomo, G.D., Lesperance, Y., Levesque, H.: Congolog, a concurrent programming language based on the situation calculus. Artificial Intelligence **121**(1-2) (2000) 109–169
8. Parr, R., Russell, S.: Reinforcement learning with hierarchies of machines. In: Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10. (1997) 1043–1049
9. Andre, D., Russell, S.: Programmable reinforcement learning agents. In: Proceedings of the 2001 Conference on Advances in Neural Information Processing Systems 13. (2000) 1019–1025
10. Dietterich, T.: The MAXQ method for hierarchical reinforcement learning. In: Proceedings of the Fifteenth International Conference on Machine Learning. (July 1998) 118–126

Towards Dependency Semantics for Conflict Handling in Logic Programs

Patrick Krümpelmann

Information Engineering Group
Technische Universität Dortmund
Germany
patrick.kruempelmann@udo.edu

Abstract. We present a dependency framework for the definition of semantics for non conflict free non-monotonic belief bases represented by extended logic programs. In particular, we define general tools for handling conflicts in such belief bases leading to a modular framework for the specification of possible ways to handle conflicts. Furthermore, we make use of these possibilities and define concrete instantiations, showing relations to other approaches. Based on this we present ways to improve these by means of changes to modules of the framework which lead to the definition of improved approaches to conflict handling in logic programming based knowledge bases.

1 Introduction

This work deals with dynamic knowledge bases which are represented by non-monotonic formalisms and in particular by extended logic programs. Mechanisms for treating dynamic knowledge bases are needed in many areas, with multiagent systems being one very prominent, and dominant, area. In a multiagent system an agent interacts with its environment as well as with other agents in the system and thus unavoidably acquaints new information during runtime which is likely to lead to conflicting beliefs which has to be dealt with in a sensible manner. Logic programming is intensively used for knowledge representation and proofed to play an important role for the development of intelligent systems and advanced reasoning tasks in those, cf. [1].

In this paper we define a general framework of dependencies for extended logic programs and sequences of these in the spirit of [2] and [3,4] which we will then use to analyse and compare common approaches, and to find improvements of these. This framework is intended to be based on logic programming, but to detach from the syntactic approach towards a semantic view of conflict handling in the dynamics of logic programming. Thus, this framework provides dependency based semantics for logic programs and sequences of these, including conflicts and the solution of these. This semantics can be used to acquaint more in-depth insights into the detection and elimination of conflicts as it gives a formal definition of these methods. We construct the framework in a modular way by making key definitions variable. Hereby, several instantiations of this

framework can be defined for means of comparison, analysis and for different scenarios of application. In this work, we are going to start elaborating on the properties and possibilities of this kind of semantics and show similarities and differences with a big class of current approaches. We also show that this can lead to deeper insights and to the definition of more powerful ways of treating dynamics of beliefs. This formal representation can, and is intended to, find new approaches to conflict handling and provides the means for general and powerful operations, incorporating many aspects of dynamics in non-monotonic logics. First results in this direction are presented in this paper.

Section 2 gives some preliminary notations of extended logic programs and program sequences. In Section 3 we will define a general dependency framework for inconsistent extended logic programs and program sequences including semantics for these. Section 4 describes and analyses different instantiations of the framework presenting new approaches. Section 5 gives a brief discussion of the presented work.

2 Preliminaries

An extended logic program consists of rules over a set of atoms using strong negation $\neg$ and default negation not. A literal L can be an atom A or a negated atom $\neg A$. The complement of a literal L is denoted by $\neg L$ and is A if $L = \neg A$ and $\neg A$ if $L = A$. Let $\mathcal{A}$ be the set of all atoms and Lit the set of all literals $Lit = \mathcal{A} \cup \{\neg A \,|\, A \in \mathcal{A}\}$. $\mathcal{D} = \{\mathsf{not}\ L \,|\, L \in Lit\}$ denotes the set of all default negated literals. And $\xi = Lit \cup \mathcal{D}$ represents the set of all literals and default negated literals. A rule r is written as

$$L \leftarrow L_0, \ldots, L_m, \mathsf{not}\ L_{m+1}, \ldots, \mathsf{not}\ L_n.$$

where the head of the rule $L = H(r)$ is either empty or consists of a single literal and the body $\mathcal{B}(r) = \{L_0, \ldots, L_m, \mathsf{not}\ L_{m+1}, \ldots, \mathsf{not}\ L_n\}$ is a subset of ξ. The body consists of a set of literals $\mathcal{B}(r)^+ = \{L_0, \ldots, L_m\}$ and a set of default negated literals denoted by $\mathcal{B}(r)^- = \{\mathsf{not}\ L_{m+1}, \ldots, \mathsf{not}\ L_n\}$. Given this we can write a rule as

$$H(r) \leftarrow \mathcal{B}(r)^+, \mathcal{B}(r)^-.$$

If $\mathcal{B}(r) = \emptyset$ we call r a fact. A set of literals which is consistent, *i. e.,* it does not contain complementary literals L and $\neg L$, is called an state I. A literal L is true in I iff $L \in I$ and false otherwise. The body $\mathcal{B}(r)$ of a given rule r is true in I iff each $L \in \mathcal{B}(r)^+$ is true in I and each $L \in \mathcal{B}(r)^-$ is false in I. A rule r is true in I iff $H(r)$ is true in I whenever $\mathcal{B}(r)$ is true in I. An state is a model of a program P if r is true in I for all $r \in P$. The reduct P^S of a program P relative to a set S of literals is defined as:

$$P^S = \{H(r) \leftarrow \mathcal{B}^+(r) \mid r \in P, \mathcal{B}^-(r) \cap S = \emptyset\}.$$

An answer set of a program P is an state I which is a minimal model of P^I.

In the sense of dynamics in logic programs, sequences of these are used to structure the knowledge base. The agent is assumed to receive information in the form of logic programs which can, for example, be seen as an update of the current knowledge of the agent [5,6] or as information from different sources [7]. The agent thus accumulates a sequence of logic programs which constitutes its knowledge base. The belief base of an agent consists therefore of a sequence of programs as defined next.

Definition 1 (Program sequence). *A* program sequence *is a sequence of logic programs $\mathcal{P} = (P_1, \ldots, P_n)$ in which each P_i is preferred over all programs in the preceding sequence of programs $(P_1, \ldots, P_{i-1})$ according to some total preference relation $<$ on the set of programs.*

We use the set notation also for sequences for the sake of readability, e. g. it can be written $P_i \in \mathcal{P}$ or $\mathcal{P} \subseteq \mathcal{P}'$. A nonmonotonic knowledge base represented as a program sequence needs the definition of some semantics. By defining a framework for program sequences we acquire such an semantics as will be laid out in the next section.

3 General Framework

In the following we will develop a general framework for the representation of dependencies in extended logic programs and in sequences of these. In particular, the possibilities for the resolution of conflicts shall be investigated in comparison to some existing approaches. For this means we will introduce identifiers for some definitions, turning these into modules of the framework that will be exchanged in the following section in order to create different instantiations of the framework which will be given by tuples of identifiers. We start by giving the necessary formalisms for dependencies in logic programs which lead to the definition of dependency semantics for program sequences which will later be analysed and extended.

Non-monotonic formalisms are characterized by their use of assumptions in what is called a default rule. In extended logic programs these are given in the form of the negative body of a rule. These assumptions create dependencies as other literals are inferred based on these which we will formalize in the following.

Definition 2 (Dependency relation). *A* dependency relation $\mathcal{R}$ *is a set of dependencies which are tuples of the form $(L, \mathcal{W})$, where the* dependant $L \in Lit$ *depends on a set of assumptions $\mathcal{W} \subseteq \xi$, also called the* premise *of the dependency.*

The dependency of a literal of some, possibly default negated, literals means that the latter have to be known or assumed in order to infer the former. The rules of the logic program resemble basic dependencies which generate further dependencies by means of chaining of rules. In order to reflect this generation of dependencies we introduce the notion of dependency sequences which are chains of sequencial applicable dependencies.

Definition 3 (Dependency sequence). *A dependency sequence consisting of dependencies* $\{d_1, \ldots, d_n\} \subseteq \mathcal{R}$ *with* $d_i = (L_i, \mathcal{W}_i)$ *is of the form:*

$$\sigma = (d_1, \ldots, d_n),\ n > 1.$$

A dependency sequence σ *is called generating for a dependency* $d = (L, \mathcal{W})$*, denoted by* σ_d *in a dependency relation* $\mathcal{R}$ *iff it satisfies the following conditions:*

(i) $d_n = (L, \mathcal{W}_n)$
(ii) $\mathcal{W} \supseteq \mathcal{W}_1$
(iii) *For each* $i, 1 \leq i < n, \mathcal{W} \cup \{L_1, \ldots, L_i\} \supseteq \mathcal{W}_{i+1}$
(iv) $\mathsf{not}\ L \notin \mathcal{W} \setminus \bigcup_{1 \leq i < n} \mathcal{W}_i$

The set of all generating dependency sequences for a dependency d is denoted by Θ_d*.*

Based on the notion of dependency sequences we can define a closure operator which can be seen as a combination of the transitive closure and the closure under valid assumptions.

Definition 4 (Closure operator based on sequences).

$$Cl(\mathcal{R}) = \mathcal{R} \cup\ \{(L, \mathcal{W}) \mid \exists d = (L, \mathcal{W}) \wedge \sigma_d = (d_1, \ldots, d_n) \subseteq \mathcal{R}\}$$

This closure operator can be used to generate the dependency relations for extended logic programs as follows.

Definition 5. *For an extended logic program* P *we define dependencies for rules. The dependency relation for* P *is then defined as:*

$$\mathcal{R}_P = Cl(\{\langle H(r), \mathcal{B}^+(r) \cup \mathcal{B}^-(r)\rangle \mid r \in P\})$$

Example 1.

$$P = \{\ A \leftarrow \mathsf{not}\ B.\ B \leftarrow \mathsf{not}\ A.$$
$$C \leftarrow A. \quad D \leftarrow B.\}$$

Given this program the following dependency relation is generated for it:

$$\begin{aligned}\mathcal{R}_P = \{&(A, \{\mathsf{not}\ B\}), (B, \{\mathsf{not}\ A\}), (C, \{A\}), (D, \{B\}), (C, \{\mathsf{not}\ B\}),\\ &(D, \{\mathsf{not}\ A\}), (C, \{\mathsf{not}\ B, \mathsf{not}\ D\}), (D, \{\mathsf{not}\ A, \mathsf{not}\ C\}),\\ &(D, \{\mathsf{not}\ A, \mathsf{not}\ C, \mathsf{not}\ D\}), \ldots\}\end{aligned}$$

Some dependencies, namely those that represent the dependence of a literal from a set of assumptions or any subset of assumptions, have a key function as they give a representation of literals being dependent solely on a set of default negated literals.

As our aim here is to investigate semantics for sequences of logic programs we generalize the definition of dependency relations of single programs to program sequences.

Definition 6. *The dependency relation for a program sequence* $\mathcal{P} = (P_1, \ldots, P_n)$ *is given by* $\mathcal{R}_{\mathcal{P}} = \mathcal{R}_{\cup P_i \in \mathcal{P}}$.

This definition treats the sequences as one big program, the structure of the sequence will be captured by a preference relation on dependencies to be introduced later.

Alternative models for a default theory are generated by different sets of assumptions which satisfy certain properties. These models can therefore be characterised using sets of assumptions which, by use of a consequence relation, generate the models of the theory.

Definition 7 (Consequence operator). *Given a dependency relation* $\mathcal{R}$ *and a set of assumptions* Δ *the set of consequences for these is given by the following operator:*

$$Cn_{\mathcal{R}}(\Delta) = \{L \mid (L, \delta) \in \mathcal{R}, \delta \subseteq \Delta\}$$

Sets of assumptions have to satisfy some conditions in order to be considered a valid characterization of the model of the theory. These are, generally speaking, consistency and maximality and are specified in the following definition. We will show later how this corresponds to answer sets of extended logic programs.

Definition 8 (Valid sets of assumptions). *A set of assumptions* Δ *is called a* valid set of assumptions *with respect to a dependency relation* $\mathcal{R}$ *iff its set of consequence is consistent,* $\{L, \overline{L}\} \notin Cn_{\mathcal{R}}(\Delta)$ *for no* $L \in Lit$*, and there exists no set of assumptions* Δ' *with* $\Delta \subset \Delta'$ *and* $\{L, \overline{L}\} \notin Cn_{\mathcal{R}}(\Delta')$ *for no* $L \in Lit$.

As an example for valid sets of assumptions consider the following program.

Example 2.

$$P = \{B \leftarrow \mathsf{not}\ A., \quad A \leftarrow \mathsf{not}\ B.\}$$

In this example there are two valid sets of assumptions, namely $\Delta_1 = \{\mathsf{not}\ B, \mathsf{not}\ \neg B, \mathsf{not}\ \neg A\}$ and $\Delta_2 = \{\mathsf{not}\ A, \mathsf{not}\ \neg A, \mathsf{not}\ \neg B\}$. The according sets of consequences are $Cn_{\mathcal{R}_P}(\Delta_1) = \{A\}$ and $Cn_{\mathcal{R}_P}(\Delta_2) = \{B\}$ which are also the only two answer sets of this program.

Consistency of information cannot be granted in many scenarios such as dynamic information and conflicts are likely to arise. These are formalised as follows in the dependency framework.

Definition 9 (Conflicts). *A dependency relation* $\mathcal{R}$ *contains a* conflict $C \subseteq \mathcal{R}$ *iff for some* $L \in Lit$ *and some* $\mathcal{W} \subseteq \mathcal{D}$ *it is the case that* $C = \{(L, \mathcal{W}), (\neg L, \mathcal{W})\}$.

Example 3.

$$P_1 = \{\ C \leftarrow \mathsf{not}\ A.,\ B \leftarrow C.\}$$
$$P_2 = \{\ \neg B \leftarrow \mathsf{not}\ A.\ \}$$

The union of the programs of this sequence is a program and generates the dependency relation $\mathcal{R}_{P_1 \cup P_2}$. The only conflict in $\mathcal{R}_{P_1 \cup P_2}$ is $C = \{(B, \{\mathsf{not}\ A\}), (\neg B, \{\mathsf{not}\ A\})\}$.

For the solution of conflicts some information, or in this case dependencies, have to be given up which leads to the formalisation of incision sets as follows.

Definition 10 (Incision $\langle I \rangle$). *Let $\mathcal{R}$ be a dependency relation and C a conflict. A set of dependencies $I \subseteq \mathcal{R}$ is called an incision of a conflict C iff*

$$C \not\subseteq Cl(\mathcal{R} \setminus I)$$

and no I' satisfying the property above with $I' \subset I$ exists.

Note that this definition is not equivalent to the condition $C \cap I \neq \emptyset$, which is necessary but not sufficient.

Example 4. Looking at the conflict C of Example 3 we get the incisions $I_1 = \{(B, \{\mathsf{not}\ A\}), (B, \{C\})\}$, $I_2 = \{(B, \{\mathsf{not}\ A\}), (C, \{\mathsf{not}\ A\})\}$ and $I_3 = \{(\neg B, \{\mathsf{not}\ A\})\}$

Given that the preferences on program sequences are given by the order of the programs, we can formalize the following preference criterion for the generated dependencies.

Definition 11 (Preference on dependencies $\langle \prec_P^d \rangle$). *Given the programs P and P'. Let d and d' be two dependencies with $d \in \mathcal{R}_P$ and $d' \in \mathcal{R}_{P'}$. Then $d' \prec d$ iff $P' \prec P$.*

The preference relation on dependencies defined this way does not generalize to dependencies $d \in \mathcal{R}_{\cup_{P_i \in \mathcal{P}}} \setminus \bigcup_{P_i \in \mathcal{P}} \mathcal{R}_{P_i}$, because these dependencies are not generated by one single rule but by means of rules from multiple programs. Other definitions of preferences that overcome this restriction are feasible and will be discussed later. Here, we will solve conflicts based on these dependencies using a base representation of the dependency relation which also has the advantage of being more concise. We will introduce a base representation of dependency relations in the following.

Definition 12 (Base Dependency). *A dependency $d \in \mathcal{R}$ is called a base dependency of the dependency relation $\mathcal{R}$ iff*

$$Cl(\mathcal{R} \setminus d) \neq Cl(\mathcal{R}).$$

A base relation $\mathcal{R}^b$ for a dependency relation $\mathcal{R}$ is a minimal set of base dependencies such that $Cl(\mathcal{R}^b) = Cl(\mathcal{R})$. A dependency sequence entirely consisting of base dependencies is called a base sequence $(d_1, \ldots, d_n),\ d_i \in \mathcal{R}^b,\ 1 \leq, i, \leq n$. The set of all base dependencies is denoted by Θ^b and the set of all base dependencies for a dependency d by Θ_d^b.

Proposition 1. *For every dependency d which has a finite generating dependency sequence σ_d there exists a generating base sequence σ_d^b for d.*

Proof. Each non base dependency d' of a dependency sequence σ can be replaced by a generating dependency sequence $\sigma^{d'}$ representing d' such that a new dependency sequence σ' is attained. Hence, any generating dependency sequence can be turned into a base sequence generating the same dependency.

Corollary 1. *A base dependency d^b is atomic in the sense that $\Theta_d^b = \{(d^b)\}$ holds, i. e., it is only generated by one atomic sequence which consists of itself.*

Proof. Assume a dependency relation $\mathcal{R}$ contains a sequence $\sigma = (d_1, \ldots, d_n)$ representing the base dependency d. Then $Cl(\mathcal{R} \setminus d) = Cl(\mathcal{R})$ as σ generates d; thus, d is not a base dependency.

As an example for the base representation of dependency relations consider the following program and the generated dependency relation.

Example 5.

$$P = \{B. \quad \neg A. \quad A \leftarrow B.\}$$

The dependency relation generated by this program $\mathcal{R}_P$ is represented by the following base dependencies.

$$\mathcal{R}_P^b = \{(B, \emptyset), (\neg A, \emptyset), (A, \{B\})\}$$

These three dependencies clearly correspond to the three rules in the program. In the closure of these dependencies the transitivity of the dependencies leads to the dependency $(A, \emptyset)$. The premise closure would add the following dependencies: $(B, \{\mathsf{not}\ A\})$, $(B, \{\mathsf{not}\ \neg A\})$, $(B, \{\mathsf{not}\ A, \mathsf{not}\ \neg A\})$, ...

Incisions defined in the dependency framework solve conflicts on a subset of the dependency relation of the program sequence as $\bigcup_{P_i \in \mathcal{P}} \mathcal{R}_{P_i} \subseteq \mathcal{R}_{\mathcal{P}}$. Due to the closure operator conflicts are reinstated in the base representation. To avoid this, incisions need to be generalised in order to solve conflicts in $\mathcal{R}_{\mathcal{P}}$ as follows.

Definition 13 (Incision closure). *Let $\mathcal{P}$ be a program sequence and $C \in \mathcal{R}$ a conflict of $\mathcal{P}$. Given an incision $I \subseteq Q$ on a domain $Q \subseteq Cl(\mathcal{R})$ such that $C \not\subseteq Cl(Q \setminus I)$ then*

$$I_{Cl} = \mathcal{R}_P \setminus Cl\,(Q \setminus I)$$

Base dependencies are atomic such that there is only the trivial base sequence that expresses d^b. These are invalidated, *i. e.,* removed, by simply removing d from $\mathcal{R}_{\mathcal{P}}$. In order to invalidate a non atomic sequences σ one of the dependencies $d' \in \sigma$ has to be invalidated in $\mathcal{R}_{\mathcal{P}}$. This underlies the next definition of base incisions for conflicts as every dependency can be invalidated by invalidating base dependencies in the end.

Definition 14 (Base incision). *A set of base dependencies I^b is called a* base incision *of a conflict C iff*

$$C \not\subseteq Cl\left(\mathcal{R}_{\mathcal{P}}^b \setminus I^b\right)$$

Example 6. For Example 3 we get the base incisions $I_1^b = \{(B, \{C\})\}$, $I_2^b = \{(C, \{\mathsf{not}\ A\})\}$ and $I_3^b = \{(\neg B, \{\mathsf{not}\ A\})\}$

We can extend base incisions to general incisions in the following way.

Definition 15 (Base generated incision). *Following Definition 13 the incision generated by a base incision for a dependency relation $\mathcal{R}_P$ is defined as:*

$$I_{Cl} = \mathcal{R}_\mathcal{P} \setminus Cl\left(\mathcal{R}_\mathcal{P}^b \setminus I^b\right)$$

Example 7. The incisions of Example 4 are generated by the base incisions of Example 6.

The incisions constructed this way are minimal since removing any single element from the incision would reinstate the conflict. Among these minimal incisions the notion of preference is applied in order to select the most preferred incisions that lead to consistent subsets of the dependency relation.

With sequences of programs, a linear ordering is granted. For the notion of preferences of sequences under the assumption of a linear ordering the following observations can be made. If the order on the programs of $\mathcal{P}$ is linear then the order on the base dependencies $\mathcal{R}_\mathcal{P}^b$ is linear as well. This does not hold for $\mathcal{R}_\mathcal{P}$ in general, as any $d \in \mathcal{R}_\mathcal{P} \setminus \bigcup_{P_i \in \mathcal{P}} \mathcal{R}_{P_i}$ is incomparable to any other $d' \in \mathcal{R}_\mathcal{P}$. For the comparison of incisions, which are sets of dependencies, incision sequences are defined in the following.

Definition 16 (Incision sequence). *An* incision sequence *is of the form $\Gamma_I = (d_1, \ldots, d_n)$ and is generated by a set of dependencies $I = \{d_1, \ldots, d_n\}$ with an ordering relation $\prec$ on the dependencies, such that $d_i \succeq d_{i+1}$, $1 < i \leq n$.*

The definition of incision sequences gives means to define preference criteria on incisions more intuitively which we will use in the following definition.

Definition 17 (Incision preference $\langle \prec_{lex}^I \rangle$). *Given two incisions I and I' of the conflict C and their corresponding incision sequences $\Gamma_I = (d_1, \ldots, d_n)$ and $\Gamma_{I\prime} = (d'_1, \ldots, d'_l)$, then $I \prec I'$ holds iff one of the following conditions holds:*

(i) $n < l \ \wedge \ \forall i, 1 \leq i \leq n : d_i = d'_i$
(ii) $n = l \ \wedge \ \exists j, 1 \leq j \leq n : d_j \prec d'_j \wedge \ \forall i, 1 \leq i \leq j : d_i \preceq d'_i$

It can be argued that this approach to preferences on incisions should be improved in order to realize more reasonable operations on program sequences but for the means of this paper we will stick to this simpler, and yet powerful, definition. Being able to express preferences on incisions leads to the notion of preferred incisions.

Definition 18 (Preferred incision). *An incision $\mathcal{I}$ of a conflict C is called a* preferred incision *iff there is no incision $\mathcal{I}'$ which is preferred over $\mathcal{I}$.*

Preferred incisions induce a selection of a subset of the dependency relation which is consistent and can be used for a semantic characterisation by resolving conflicts using preferred incisions as defined in the following.

Definition 19 (Consistent subset). *Given a dependency relation $\mathcal{R}$ that contains a set of conflicts $\mathbb{C} = \{C_1, \ldots, C_n\}$. We call a subset $\lessdot \subseteq \mathcal{R}$ of the dependency relation consistent iff it has a valid set of assumptions and is constructible as $\lessdot = \mathcal{R} \setminus \mathbb{I}$ where:*

$$\mathbb{I} \in \{\bigcup_{1 \leq i \leq n} \mathcal{I}_i \mid \mathcal{I}_i \text{ is a preferred incision of the conflict } C_i \in \mathbb{C}\}$$

The definition of incisions and the application of these solves conflicts and leads to the definition of semantics for the underlying theory which is given in as follows.

Definition 20 (Dependency semantics). *The set of* extensions *E of a dependency relation $\mathcal{R}$ are given by a set of pairs of the form $(\Delta, \lessdot)$, where $\lessdot$ is a consistent subset of $\mathcal{R}$ and Δ is a valid set of assumptions for $\lessdot$.*

For consistent programs we can show that the defined dependency semantics is sound and complete as has been indicated in Example 2 and formalized in the following theorem.

Theorem 1. *A set of assumptions $\Delta \subseteq \mathcal{D}$ is valid in a dependency relation generated by a consistent extended logic program P $\mathcal{R}_P$ iff $Cn_{\mathcal{R}}(\Delta)$ is an answer set of P. Let S be an answer set of P, then there exists a valid set of assumptions $\Delta \subseteq \mathcal{D}$ w. r. t. $\mathcal{R}_P$ such that $S = Cn_{\mathcal{R}_P}(\Delta)$.*

Proof. Δ is valid iff $Cn_{\mathcal{R}}(\Delta)$ consistent and set inclusion maximal in this property. Thus for each dependency $(L, \mathcal{W})$ with $\mathcal{W} \subseteq \Delta$ it holds that $L \in Cn_{\mathcal{R}}(\Delta)$. By the consistency condition it holds that $\Delta \cap Cn_{\mathcal{R}}(\Delta) = \emptyset$ and no dependency generated by a rule $r \notin P^{Cn_{\mathcal{R}}(\Delta)}$ is relevant. $Cn_{\mathcal{R}}(\Delta)$ contains therefore all literals which can be inferred by means of consistent assumptions and no more. Hence $Cn_{\mathcal{R}}(\Delta)$ is minimal model of $P^{Cn_{\mathcal{R}}(\Delta)}$. If $Cn_{\mathcal{R}}(\Delta)$ is an answer set for P then $Cn_{\mathcal{R}}(\Delta)$ is consistent by definition. If Δ would not be maximal then $Cn_{\mathcal{R}}(\Delta)$ would not be a model of $P^{Cn_{\mathcal{R}}(\Delta)}$ as all default negation is eliminated by the reduct. If S is an answer set of P then it holds that $S = Cn_{\mathcal{R}_P}(\Delta)$ with the valid set of assumptions $\Delta = \{\mathsf{not}\ L \mid L \in Lit \setminus S\}$.

4 Analysis

In the last section we have defined a set of tools, a framework and the semantics for a dependency relation generated by sequences of extended logic programs. Here, we will use the possibilities of our framework to show the differences of approaches for handling conflicts in program sequences. To this end, we will change some previous definitions that have been defined as modules and which can be exchanged in order to achieve different behaviours. By this, we are able to show relations to other approaches and to show ways to improve these.

4.1 Causal Rejection

We start to look into possible preference criteria and, at first, to look into relations to the common causal rejection principle used in many approaches for dealing with sequences of logic programs like the dynamic programming approach of Alferes et al. [6,8] as well as the inheritance programs of Buccafurri et al. [9], the ordered logic programs of Buccafurri et al. [10] and the update programs of Eiter et al. [5]. The basics underlying the approaches of causal rejection are that conflicting rules are identified and for each pair of these on is rejected, or blocked from application, such that the conflict is resolved. Thus, conflicts in programs sequences can be dealt with such that a single consistent program can be generated. Here, a basic formalisation of this follows.

Definition 21 (Conflict between rules). *A conflict is a set of two rules r, r' which have complementary head literals. The conflict between two rule r and r' is denoted by $r \bowtie r'$.*

Definition 22 (Causal rejection principle (according to [5])[1]). *The* causal rejection principle *states that a rule r is rejected iff there is another rule r' which is conflicting with r, i. e., $r \bowtie r'$, is not rejected itself and is preferred over r.*

A rule r could, for instance, be preferred over r' if it represents more recent information. In this case only rule r' is applied and rule r is discarded, resolving the conflict between both rules by observing temporal aspects. In this work the approach of update programs [5] is used as a representative for update approaches based on causal rejection. These approaches solve conflicts by rejection of one rule which directly causes the conflict, *i. e.,* its head literal is in conflict. We call such a rule a root rule. In terms of the dependency framework this means that only dependencies whose dependant is equal to one of the dependants of the dependencies in the conflict can be part of an incision. Therefore, we use the definition of an incision as module and refer to the original definition as $\langle I \rangle$ while we state a new definition $\langle I_{\text{root}} \rangle$ next.

Definition 23 (Incision $\langle I_{\text{root}} \rangle$). *An incision I to a conflict $C = \{(L, \mathcal{W}), (\neg L, \mathcal{W})\}$ is a minimal set of dependencies such that $I = \{(L, \mathcal{W}_1, \ldots, (L, \mathcal{W}_n))\}$ or $I = \{(\neg L, \mathcal{W}_1, \ldots, (\neg L, \mathcal{W}_n))\}$.*

We illustrate this instantiation of the framework, which we identify by the tuple $\langle \prec_d^P, I_{root}, \prec_I^{lex} \rangle$, using one of the standard examples of update sequences first mentioned in [6].

Example 8. The program sequence of the example consists of two extended logic programs P_1 and P_2 with $P_1 \prec P_2$ such that we get the program sequence $\mathcal{P} = (P_1, P_2)$.

$$P_1 = \{r_1 : sleep \leftarrow \mathsf{not}\ tv\_on.\ r_2 : night \leftarrow .$$
$$r_3 : tv\_on \leftarrow .\ r_4 : watch\_tv \leftarrow tv\_on\}$$
$$P_2 = \{r_5 : \neg tv\_on \leftarrow power\_failure.\ r_6 : power\_failure \leftarrow\}$$

[1] We are aware of the different definitions of this principle [8,11,12] but do not have space to discuss these here.

The dependency relation for this sequence of programs is generated as follows:

$$\mathcal{R}_{\mathcal{P}} = Cl(\ \{(sleep, \{\mathsf{not}\ tv\_on\}), (night, \emptyset), (tv\_on, \emptyset), (watch\_tv, \{tv\_on\}),$$
$$(\neg tv\_on, \{power\_failure\}), (power\_failure, \emptyset)\})$$

In this dependency relation the following conflict exists: $C = \{(tv\_on, \emptyset), (\neg tv\_on, \emptyset)\}$. The only preferred base incision for this conflict is given as: $I^b = \{(tv\_on, \emptyset)\}$ From this incision we get the extension $E = \{(\emptyset, \lessdot)\}$ with $\lessdot = R_{\mathcal{P}} \backslash I^b_{Cl}$. The consequences are given by $Cn_{\lessdot}(\emptyset) = \{power\_failure,\ \neg tv\_on,\ sleep, night\}$ which corresponds to the resulting answer set for all semantics for update sequences based on causal rejection.

Another example we want to give is one which involves an update by a tautology. Updates of this type are often discussed in the literature [5,3,12] and they seem to be a major problem semantics of update sequences. Tautologies in logic programs are rules of the type $L \leftarrow \mathcal{B}^+$. with $L \in \mathcal{B}^+$. In these the head-literal dependes on itself. A more general class are a set of rule which mediate this self dependence like in this example $P = \{A \leftarrow B., B \leftarrow A\}$ which we call a self depending program. In terms of our framework this means that $(L, \mathcal{W}) \in \mathcal{R}_P$ with $L \in \mathcal{W}$. When an update sequence is extended by a self depending program the semantics is not supposed to change as the following example, adapted from [12] [2] , demonstrates.

Example 9.

$$P_1 = \{\ day \leftarrow \mathsf{not}\ night. \quad night \leftarrow \mathsf{not}\ day.$$
$$stars \leftarrow night, \mathsf{not}\ cloudy. \quad \neg stars.$$
$$P_2 = \{\quad stars \leftarrow venus. \quad venus \leftarrow stars.$$

The first rule of the second program in this sequence is clearly self-dependent. The only answer set of the first program alone is given by $\{day, \neg stars\}$. Considering the program $P_1 \cup P_2$ we notice that it has the same set of answer sets and, in particular, is free of conflicts. Now, according to many semantics of causal rejection [6,8,9,5,10] there are two answer sets for the this sequence of programs, namely the one given above and the answer set $\{night, stars, venus\}$ which is an undesired behaviour for such semantics as there is no justification for the addition of an answer set.

In terms of the dependency semantics we can make the following observations. In this dependency relation no conflict exists as the conflicting literals $stars$ and $\neg stars$ depend on different sets of assumptions, namely we have $\{(stars, \{\mathsf{not}\ cloudy, \mathsf{not}\ day\}), (\neg stars, \emptyset)\} \subseteq \mathcal{R}_{\mathcal{P}}$. Obviously, $\{\mathsf{not}\ cloudy, \mathsf{not}\ day\}$ is a superset of $\emptyset$ and therefore $\{\mathsf{not}\ cloudy, \mathsf{not}\ day\}$ is not a valid set of assumptions as $\{stars, \neg stars\} \subseteq Cn_{\mathcal{R}_{\mathcal{P}}}$. The dependency extension of $\mathcal{P}$ is thus given by $(\mathcal{R}_{\mathcal{P}}, \{\mathsf{not}\ night\})$ and the consequences of this are $\{day, \neg stars\}$.

[2] The example has been formulated as an extended logic program instead of a general logic program here.

Thus, the dependency semantics of this instantiation does not show the undesired behaviours of many semantics of causal rejection which is a good starting point for further improvements. As a next step, we investigate another example for which the approaches based on causal rejection show an undesired behaviour.

Example 10. The following program sequence is considered $\mathcal{P} = (P_1, P_2, P_3)$.

$$P_1 = \{r_1 : B.\} \quad P_2 = \{r_2 : \neg A.\} \quad P_3 = \{r_3 : A \leftarrow B.\}$$

In this small example, three programs with different priorities, each of which is consisting of one single rule, are given. In a multiagent system the information might have been received from three different agents with different credibilities which gives rise to the prioritisation of the programs. The most preferred program could as well represent generic knowledge of this particular agent, which is entrenched most. The resulting answer set according to the causal rejection principle is $\{B, A\}$. The resulting answer set shows that the second rule has been rejected. That is, because there is a conflict, $\{A, \neg A\}$, in $\mathcal{P}$ and as $(A \leftarrow B)$ is higher prioritised as $(\neg A)$ the causal rejection principle states that the latter rule is to be rejected. This leads to a consistent answer set with respect to the given rule-priorities.

This example of a program sequence handled by the approach of update programs resolves the existing conflict. However, having a closer look at this example, it can be noticed that the belief A is part of the answer set, which in turn is solely based on B, given the rule $(A \leftarrow B)$. The latter rule comes from the highest prioritised program P_3 and thus causes the rejection of $(\neg A)$. Taking into consideration that it is the fact B which, by means of the rule from P_3, is responsible for the inference of A and that B is less preferred than $(\neg A)$ it seems to be counter intuitive that $(\neg A)$ is rejected. The rule r_3 is able to defeat r_2, since the priority of r_3 is fixed and higher than the priority of any other rule regardless of the actual instantiation, *i. e.,* the priorities of the body literals, of the rule. In opposition to this behaviour it would be more intuitive to opt for $(\neg A)$ to be preferred over A since the only reason A is believed in is B which again, is less preferred than $(\neg A)$.

In this scenario, the body of a rule is satisfied by information that is less credible than the rule itself, the inferred information receives the high credibility of the rule and the less credible base of the inference is hidden. The described behaviour is the reason for unwanted and unintuitive results in update programs and in the classic application of the causal rejection principle in general. This is due to the fact that no priority information is used in the body of the rules which could be used to evaluate the credibility of the literal that is inferred by means of the rule. The facilitation of local, or static, prioritisation of information is a major drawback in a multiagent setting. Now we are looking what the dependency semantics in this instantiation is. The dependency relation is generated by:

$$\mathcal{R} = Cl(\{(B, \emptyset), (\neg A, \emptyset), (A, \{B\}), (A, \emptyset)\})$$

In this dependency relation the following conflict exists: $C = \{(\neg A, \emptyset),\ (A, \emptyset)\}$. Possible base incisions for this conflict are: $I_a = \{(B, \emptyset)\}$, $I_b = \{(\neg A, \emptyset)\}$ and

$I_c = \{(A, B)\}$. Here, I_b is the preferred incision which results in the extension $E = \{(\emptyset, \mathcal{R}_{\mathcal{P}_2} \setminus Cl_I(I_b^b)\}$

The consequences of these are given by $Cn_{\preccurlyeq}(\emptyset) = \{A, B\}$. This corresponds to the result of semantics based on causal rejection principle which as argued above is not desired in this case. Our framework now allows us to consider modifications to different parts of this instantiation for update sequences in order to overcome this behaviour, as we will show in the following.

4.2 Improved semantics

The problems that were encountered before are due to the static treatment of rule priorities. Within our dependency framework this means that the preferences of conflicting dependencies have to be evaluated in a bit more advanced fashion. As stated before, Definition 11 does define preferences for a subset of the dependencies and this in a very simple fashion. While this seems to be sufficient to capture the causal rejection principle it also leaves space for more elaborate definitions of dependency preferences. As shown above, there is motivation to explore these, which has also been put forward in [13].

Particularly, in the dependency framework the dependencies that are part of the conflict C do not have any preference assigned to them in general as they are likely to be part of the closure. But exactly these are the important dependencies. The preference relation on dependencies Definition 11, does not extend to closure dependencies as discussed earlier. Hence, we create another module $\langle \prec \rangle$, call the previous definition $\langle \prec_P^d \rangle$ and give a new module and instantiation in the following.

Definition 24 (Preference on dependencies $\langle \prec_\diamond^d \rangle$). *For a base dependency $\sigma^b = (d_1, \ldots, d_n) \subseteq \mathcal{R}^b$ $d_i \in \sigma^b$ is the least element $min(\sigma)$ of σ iff there is no $d_j \in \sigma$ such that $j \neq i$, $d_i \in P \prec P' \ni d_j$, $P \neq P'$ and no $d_k \in \sigma$ with $k < i$, $d_i \in P$ and $d_k \in P$.*

Given a program sequence $\mathcal{P}$ and a preference relation $\mathcal{R}_\mathcal{P}$ generated by $\mathcal{P}$. A dependency $d \in \mathcal{R}_\mathcal{P}$ is preferred over another dependency $d' \in \mathcal{R}_\mathcal{P}$ iff there is a base sequence σ_d^b which generates d and for each base sequence $\sigma_{d'}^b$ that generates d' it holds that $min(\sigma_{d'}^b) \prec min(\sigma_d^b)$.

This preference criterion on dependencies determines the preference for a dependency based on its generating dependency sequences. Within this, the preference is dependent on the least preferred dependency which is part of the generation of the dependency under consideration. Moreover, all dependency sequences for the dependency are considered and the existence of one more preferred generation of a dependency over another dependency is sufficient to give preference to this. This amounts to the consideration of the maximally preferred generation of each dependency. This method of preference evaluation in logic programs can also be found in credibility logic programs [13].

In order to adapt the dependency framework, the selection of incisions has to be modified according to this preference relation on dependencies. This is the

case because the preference on closure dependencies reflects the actual credibility of the dependency while the single items used to create this dependency still have fixed, and possibly higher, credibilities. These have to be removed as well and would make the incision less preferred if not. In order to evaluate an incision using the closure credibility which is less or equally preferred as any other dependency the following preference on incision is used.

Definition 25 (Incision preference $\prec_I^1$). *An incision I for a conflict C is preferred over an incision I' denoted $I \prec I'$ iff for their corresponding incision sequences $\Gamma_I, \Gamma_{I'}$ it holds that $d_1 \prec d_1'$.*

Example 11. The following sequence $\mathcal{P} = \{P_1, P_2, P_3\}$ is considered.

$$P_1 = \{r_1 : B.\} \quad P_2 = \{r_2 : \neg A.\} \quad P_3 = \{r_3 : A \leftarrow B.\}$$

Thus now we are looking what the dependency semantics in the instantiation $\langle \prec_P^d, I_{root}, \prec_I^1 \rangle$ is. The dependency relation is generated by:

$$\mathcal{R}_\mathcal{P} = Cl(\{(B, \emptyset), (\neg A, \emptyset), (A, \{B\}), (A, \emptyset)\})$$

In this dependency relation the following conflict exists: $C = \{(\neg A, \emptyset), (A, \emptyset)\}$. Possible base incisions for this conflict are: $I_a = \{(B, \emptyset)\}$, $I_b = \{(\neg A, \emptyset)\}$ and $I_c = \{(A, \{B\})\}$. In contrast to the instantiation considered before, here, I_c is the preferred incision which results in the extension $E = \{(\emptyset, \lessdot\}$ with $\lessdot = \mathcal{R}_{\mathcal{P}_2} \setminus Cl_I(I_c^b)$ The consequences of these are given by $Cn_\lessdot(\emptyset) = \{\neg A, B\}$.

The framework instantiation considered here avoids the superficial rejection of information without considering the reliability of the whole proof of this information as desired. This change of the semantics has been achieved by the change of two definitions in the framework.

5 Discussion

In this work, a versatile and comprehensive dependency framework for non-monotonic belief bases represented by sequences of extended logic programs has been introduced to define a modular semantical framework abstracting from the syntactical level. We developed the framework on a general level, introducing module parts of the framework that open possibilities of investigations of different versions and their behaviours. Based on this we showed relations to other approaches and gave different instantiations featuring advantages over the others. It has been shown how this modularisation can be used to model the behaviour of other approaches by which the semantics are made explicit in one single framework. This makes the analysis and comparison of different approaches as well as the development of new ones easier. Moreover, we presented a new way of conflict solution by means of source rejection using an instantiation of the framework.

A lot of work has been done in the field of the development of mechanism for dealing with sequences of logic programs and also on the development of dependency semantics for non-monotonic formalisms. In terms of the use of the latter for the expression of the former we are only aware of the work in [3]. This work is similar to that one in the basic approach but differs widely in the definition of the semantics, representation and application as well as in the modular representation.

Future work will clearly lie in the extension of the framework towards more features of dynamics of logic programs and in the formalisation of achieved results. This has the potential to lead to improved semantics and approaches to the handling of conflicts in non-monotonic formalisms.

References

1. Gelfond, M., Leone, N.: Logic programming and knowledge representation — the A-Prolog perspective. Artificial Intelligence **138**(1–2) (2002) 3–38
2. Bondarenko, A., Toni, F., Kowalski, R.A.: An assumption-based framework for non-monotonic reasoning. In: Proc. 2nd International Workshop on Logic Programming and Non-monotonic Reasoning, MIT Press (1993) 171–189
3. Sefranek, J.: Rethinking semantic of dynamic logic programming. In: Proceedings of the 11'th Intl. Workshop on Non-Monotonic Reasoning (NMR-06). (2006)
4. Sefránek, J.: Irrelevant updates and nonmonotonic assumptions. In Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A., eds.: Logics in Artificial Intelligence, 10th European Conference, JELIA 2006, Liverpool, UK. Volume 4160. (2006) 426–438
5. Eiter, T., Fink, M., Sabbatini, G., Tompits, H.: On properties of update sequences based on causal rejection. Theory Pract. Log. Program. **2**(6) (2002) 711–767
6. Alferes, J.J., Leite, J.A., Pereira, L.M., Przymusinska, H., Przymusinski, T.C.: Dynamic logic programming. In Cohn, A., Schubert, L., Shapiro, S., eds.: Proc. of the 6th Intl Conference on Principles of Knowledge Representation and Reasoning (KR-98), San Francisco, Morgan Kaufmann Publishers (June 2–5 1998) 98–111
7. Krümpelmann, P., Thimm, M., Ritterskamp, M., Kern-Isberner, G.: Belief operations for motivated BDI agents. In: Proc. of 7th Int. Joint Conference on Autonomous Agents and Multiagent Systems 2008 (AAMAS '08). (2008)
8. Alferes, J.J., Leite, J.A., Pereira, L.M., Przymusinska, H., Przymusinski, T.C.: Dynamic updates of non-monotonic knowledge bases. Journal of Logic Programming **45**(1-3) (2000) 43–70
9. Buccafurri, F., Faber, W., Leone, N.: Disjunctive logic programs with inheritance. In: International Conference on Logic Programming. (1999) 79–93
10. Buccafurri, F., Leone, N., Rullo, P.: Stable models and their computation for logic programming with inheritance and true negation. J. Log. Prog. **27**(1) (1996) 5–43
11. Leite, Jo a.A., Pereira, L.M.: Generalizing updates: From models to programs. In: LPKR '97: Selected papers from the 3'rd Intl. Workshop on Logic Programming and Knowledge Representation, London, UK, Springer-Verlag (1998) 224–246
12. Alferes, J., Banti, F., Brogi, A., Leite, J.: Semantics for dynamic logic programming: a principled based approach (2004)
13. Krümpelmann, P., Kern-Isberner, G.: Propagating credibility in answer set programs. In Schwarz, S., ed.: WLP08 - 22nd Workshop on (Constraint) Logic Programming Dresden, Germany. Technische Berichte, Martin-Luther-Universität Halle-Wittenberg, Germany (2008)