A System for Relational Probabilistic Reasoning on Maximum Entropy

Matthias Thimm
Department of Computer Science
Technische Universität Dortmund,
Germany

Marc Finthammer
Department of Computer Science
FernUniversität in Hagen,
Germany

Sebastian Loh
Department of Computer Science
Technische Universität Dortmund,
Germany

Gabriele Kern-Isberner
Department of Computer Science
Technische Universität Dortmund,
Germany

Christoph Beierle
Department of Computer Science
FernUniversität in Hagen,
Germany

Abstract

Comparisons of different approaches to statistical relational learning are difficult due to the variety of the available concepts and due to the absense of a common interface. The main objective of the KREATOR toolbox introduced here is to provide a common methodology for modelling, learning, and inference in a relational probabilistic framework. As a second major contribution of this paper, we present the RME approach to relational probabilistic reasoning which applies the principle of maximum entropy to groundings of a relational knowledge base and which is also supported by KREATOR.

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 (De Raedt and Kersting 2008; Getoor and Taskar 2007) for some excellent overviews. This area investigates methods for representing probabilistic information in a relational context for both reasoning and learning. Many researchers have 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 (BLPs) (Getoor and Taskar 2007, Ch. 10) and Markov logic networks (MLNs) (Getoor and Taskar 2007, Ch. 12). Other approaches also employ Bayes nets for their theoretical foundation like relational Bayesian networks (Jaeger 2002); or they are influenced by other fields of research like probabilistic relational models (Getoor et al. 2001) by database theory. There are also some few approaches on applying maximum entropy methods to the first-order case (Bacchus et al. 1996; Kern-Isberner and Lukasiewicz 2004), and, more recently, (Loh, Thimm, and Kern-Isberner 2010; Thimm 2009). 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 (Muggleton and Chen 2008). No formal knowledge representation criteria exist to date for such comparisons. Hence, applying methods to benchmark examples is an important means for the purpose of comparing and evaluating. However, even seemingly small examples need to be computed by a machine, due to the size explosion caused by grounding, and each of these approaches comes with its own computational peculiarities. What is urgently needed to advance and combine research work in this area is a system that is capable of handling different representation frameworks in parallel.

In this paper, we introduce 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[1] which aims at developing a common methodology for 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. The KREATOR system is to provide 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, KREATOR already supports BLPs and MLNs, and in particular a new approach for using maximum entropy methods in a relational context. So, as a second major contribution of this paper, we present the so-called RME approach to relational probabilistic reasoning which applies the principle of maximum entropy to groundings of the relational knowledge base. In contrast to e. g. (Bacchus et al. 1996), we assume the probabilities to be subjective, without any statistical interpretation. RME is in some sense similar to the approach pursued in (Kern-Isberner and Lukasiewicz 2004) in that instantiations of the knowledge base are used. However, we do not employ a closed world assumption, and the RME semantics is parametrized by a grounding operator which is needed to resolve conflicts. In (Kern-Isberner and Lukasiewicz 2004), conflicts are avoided by considering

[1] www.fernuni-hagen.de/wbs/research/kreate/

interval-valued probabilities, while RME is based on pointwise probabilities.

As an illustration, we will use KREATOR to compare RME with BLPs and MLNs on the following benchmark example taken from (Pearl 1998).

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?".*

The rest of this paper is organized as follows. In Sec. 2 we introduce our relational maximum entropy approach RME and give an overview on BLPs and MLNs. Sec. 3 presents the KREATOR system, Sec. 4 reports on comparing BLP, MLN and RME in the burglary example, and Sec. 5 concludes.

2. Relational Probabilistic Knowledge Representation

In this section, we briefly describe some approaches to generalize propositional probabilistic knowledge representation for the first-order context. We start by presenting a method that applies the principle of maximum entropy to a grounded version of the relational knowledge base. We continue by recalling the basics of Bayesian logic programs (Getoor and Taskar 2007, Ch. 10) and of Markov logic networks (Getoor and Taskar 2007, Ch. 12).

Relational Maximum Entropy (RME)

As a base of our first-order probabilistic language, we use a fragment of a classical first-order language without function symbols and without quantifiers. As we use a many-sorted setting, constants and variables are partitioned into disjoint sorts; note that this easily covers also the unsorted case by simply using a single sort. An RME signature $\Sigma = (S, U, Pred)$ consists of a finite set of sorts S, a finite S-indexed set of constants U, and a finite S^*-indexed set of predicates $Pred$; for $p \in Pred_{s_1 \ldots s_n}$ we write $p(s_1, \ldots, s_n)$. All formulas and all substitutions must be well-sorted in the usual sense. A grounding substitution instantiates variables with constants. The first-order language generated by Σ is denoted by $\mathcal{L}_\Sigma$ or just by $\mathcal{L}$.

The basic idea of our relational maximum entropy framework (RME) is to make use of propositional maximum entropy techniques (Kern-Isberner 1998; Rödder and Meyer 1996) after grounding the knowledge base appropriately. The *entropy* H is an information-theoretic measure on probability distributions and is defined as a weighted sum on the information encoded in every possible world $\omega \in \Omega$: $H(P) = -\sum_{\omega \in \Omega} P(\omega) \log P(\omega)$. By employing the *principle of maximum entropy* one can determine the single probability distribution that is the optimal model for a consistent knowledge base $\mathcal{R}$ in an information-theoretic sense: $P^{ME}_{\mathcal{R}} = \arg\max_{P \models \mathcal{R}} \mathcal{H}(P)$. However, this depends crucially on $\mathcal{R}$ being consistent, for otherwise no model of $\mathcal{R}$ exists, let alone models with maximum entropy. This problem is all the more difficult in a first-order context with free variables, as groundings may introduce non-trivial conflicts. We will not go into more detail here, but refer to a companion paper (Loh, Thimm, and Kern-Isberner 2010) that focusses on grounding strategies. In this paper, we will only consider knowledge bases that allow a straightforward consistent grounding for RME knowledge bases since the aim of this paper is to present the basic properties of the RME framework and compare it to other probabilistic relational logics such as MLNs and BLPs for which the problem of inconsistent grounding does not exist. To avoid conflicts and to alleviate comparisons, we restrict the language of conditional formulas and associate meta-constraints for possible groundings to them.

Definition 1 (RME conditional) *An* RME conditional $r = (\phi \mid \psi)[\alpha][c]$ *over* $\Sigma = (S, U, Pred)$ *consists of a head literal* $\phi \in \mathcal{L}$*, a list of* n *body literals* $\psi = \psi_1, \ldots, \psi_n \in \mathcal{L}$ *which is understood as the conjunction* $\psi_1 \wedge \ldots \wedge \psi_n, n \geq 0$*, 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* $\{k_1, \ldots, k_l\} \subseteq U$*. An RME conditional* r *is called* ground *iff* r *contains no variables.*

$(\mathcal{L} \mid \mathcal{L})^{RME}$ *is the set of all RME conditionals built from* $\mathcal{L}$*, and the set of all ground RME conditionals is referred to by* $(\mathcal{L} \mid \mathcal{L})^{RME}_U$*.*

Remark that although we use an underlying quantifier-free language variables appearing in conditionals are implicitly universally quantified over the whole conditional. RME knowledge bases are sets of RME conditionals, together with the basic structures of the language.

Definition 2 (RME knowledge base) *An* RME knowledge base KB *is a quadruple* $KB = (S, U, Pred, \mathcal{R})$ *with an RME signature* $\Sigma = (S, U, Pred)$ *and a finite set of RME conditionals* $\mathcal{R}$ *over* Σ*.*

For the purpose of illustration, we represent Ex. 1 as an RME knowledge base.

Example 2 *Let* KB *contain sorts* $S = \{Person, Town, Status\}$*, constants* $U_{Person} = \{james, carl\}$ *of sort* $Person$*,* $U_{Town} = \{yorkshire, austin\}$ *of sort* $Town$*,* $U_{Status} = \{bad, average, good\}$ *of sort* $Status$*, predicates* $Pred = \{alarm(Person), burglary(Person), lives\_in(Person, Town), nhood(Person, Status)\}$*, and conditionals* $\mathcal{R} = \{c_1, \ldots, c_7\}$*, as listed below:*

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

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

As sketched above, semantics are given to RME knowledge bases by grounding $\mathcal{R}$ with a *grounding operator* (GOP) $\mathcal{G} : \mathfrak{P}((\mathcal{L} \mid \mathcal{L})^{RME}) \rightarrow \mathfrak{P}((\mathcal{L} \mid \mathcal{L})_U^{RME})$ which maps knowledge bases to ground knowledge bases; here $\mathfrak{P}(S)$ denotes the power set of a set S.

Ground conditionals $r \in (\mathcal{L} \mid \mathcal{L})_U^{RME}$ can be interpreted as in the propositional case, i.e. $P \models (\phi|\psi)[\alpha]$ iff $P(\phi|\psi) = \alpha$ and $P(\psi) > 0$, where P is a probability distribution over the Herbrand base of $\mathcal{L}$, and $P(\phi) = \sum_{\omega \in \Omega, \omega \models \phi} P(\omega)$ for any classical formula ϕ. Here Ω is the set of all Herbrand interpretations of $\mathcal{L}$ and provides a possible worlds semantics for the classical part of $\mathcal{L}$. Non-conditional formulas $(\phi)[\alpha]$ with ϕ being a literal can be considered consistently as conditionals with empty premise $(\phi \mid \top)[\alpha]$, so that no explicit distinction between conditionals and flat formulas is necessary in the following.

If the ground knowledge base $\mathcal{G}(\mathcal{R})$ is consistent, $P^{ME}_{\mathcal{G}(\mathcal{R})}$ can be calculated as in the propositional case (Rödder and Meyer 1996). An RME conditional $q \in (\mathcal{L} \mid \mathcal{L})^{RME}$ is *RME entailed by the RME knowledge base $\mathcal{R}$ under the grounding $\mathcal{G}$*, in symbols

$$\mathcal{R} \models^{ME}_{\mathcal{G}} q \text{ iff } P^{ME}_{\mathcal{G}(\mathcal{R})} \models \mathcal{G}(q),$$

i. e. iff for all $q^* \in \mathcal{G}(q), P^{ME}_{\mathcal{G}(\mathcal{R})} \models q^*$.

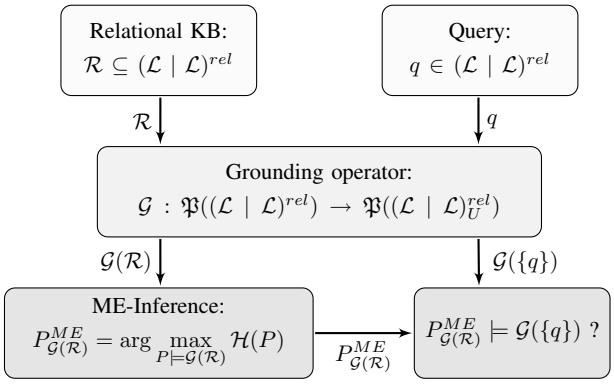

Figure 1: The RME inference process

The RME inference process can be divided into three steps (cf. Fig. 1): 1.) ground the knowledge base with a grounding operator $\mathcal{G}$, 2.) calculate the probability distribution $P^{ME}_{\mathcal{G}(\mathcal{R})}$ with maximum entropy for the grounded instance $\mathcal{G}(\mathcal{R})$, and 3.) calculate all probabilistic implications of $P^{ME}_{\mathcal{G}(\mathcal{R})}$. More details on RME semantics can be found in (Loh, Thimm, and Kern-Isberner 2010).

Example 3 *Let $KB = (S, U, Pred, \mathcal{R})$ be the RME knowledge base as described in Ex. 2. Let $Q_1 = (alarm(\mathsf{james}) \mid E_1)$ a query with the evidences $E_1 = \{$lives_in(james, austin), tornado(austin), neighborhood(james, average)$\}$.*

Initially $\mathcal{R}$ and Q_1 are grounded by a GOP $\mathcal{G}$. In this case we just use universal substitution of all occurring variables with all possible constants. The grounding of Q_1 is simply the identity $\mathcal{G}_\chi(Q_1) = Q_1$ as Q_1 does not contain any variables. Then $P^{ME}_{\mathcal{G}_\chi(\mathcal{R})}$ can be calculated and the probability of the query Q_1 can be given as

$$P^{ME}_{\mathcal{G}(\mathcal{R})}(alarm(\mathsf{james}) \mid E_1) = 0.8951 \quad .$$

Bayesian Logic Programs

Bayesian logic programming combines logic programming and Bayesian networks (Getoor and Taskar 2007, Ch. 10). In contrast to first-order logic, Bayesian logic programs (BLPs) make use of multi-valued *Bayesian predicates* from which *Bayesian atoms* can be built by using constants and variables. Each ground Bayesian atom represents a single random variable. If A is a Bayesian atom of the Bayesian predicate p, we denote the set of possible values, or *states*, that p, or A, can take by $S(p) = S(A)$.

The basic structure for knowledge representation in Bayesian logic programs are *Bayesian clauses* which model probabilistic dependencies between Bayesian atoms.

Definition 3 (Bayesian Clause) *A* Bayesian clause *c is an expression $(H \mid B_1, \ldots, B_n)$ with Bayesian atoms $H, B_1, \ldots, B_n$. To each such clause, a function $\mathsf{cpd}_c : S(H) \times S(B_1) \times \ldots \times S(B_n) \rightarrow [0,1]$ is associated such that $\sum_{h \in S(H)} \mathsf{cpd}_c(h, b_1, \ldots, b_n) = 1$. for all $b_1 \in S(B_1), \ldots, b_n \in S(B_n)$. cpd_c is called a* conditional probability distribution.

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

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

where $S(tornado/1) = S(lives\_in/2) = S(alarm) = S(burglary) = \{\mathsf{true}, \mathsf{false}\}$ and $S(nhood) = \{\mathsf{good}, \mathsf{average}, \mathsf{bad}\}$. 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 $\mathsf{cpd}_{c_2}(\mathsf{true}, \mathsf{true}, \mathsf{true}) = 0.9$, $\mathsf{cpd}_{c_2}(\mathsf{true}, \mathsf{true}, \mathsf{false}) = 0.01$, $\mathsf{cpd}_{c_2}(\mathsf{true}, \mathsf{false}, \mathsf{true}) = 0.0$, and $\mathsf{cpd}_{c_2}(\mathsf{true}, \mathsf{false}, \mathsf{false}) = 0.0$.

In order to aggregate probabilities that arise from applications of different Bayesian clauses with the same head (cf., e.g., clauses c_1 and c_2 in Ex. 4), BLPs make use of *combining rules*. A combining rule cr_p for a Bayesian predicate p/n is a function cr_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. 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 $f : S(b) \times S(a_1) \times S(a_2) \rightarrow [0,1]$. Appropriate choices for such functions are *average* or *noisy-or*, cf. (Getoor and Taskar 2007, Ch. 10).

Definition 4 (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 propositional forms, i. e. into Bayesian networks (Pearl 1998). Given a specific (finite) universe U a Bayesian network BN can be constructed by introducing a node for every ground Bayesian atom in B and computing the corresponding (joint) conditional probability distributions. For a more detailed description of Bayesian Logic Programs we refer to (Getoor and Taskar 2007, Ch. 10).

Markov Logic Networks

Markov logic (Getoor and Taskar 2007, Ch. 12) establishes a framework which combines Markov networks (Pearl 1998) with first-order logic to handle a broad area of statistical relational learning tasks. The Markov logic syntax complies with first-order logic where each formula is quantified by an additional weight value. Semantics are given to sets of Markov logic formulas by a probability distribution over propositional possible worlds that is calculated as a log-linear model over weighted ground formulas. The fundamental idea in Markov logic is that first-order formulas are not handled as hard constraints but each formula is more or less softened depending on its weight. These weights induce a kind of priority ordering on the formulas of the knowledge base that determines their respective influence on the probabilities of the log-linear model. In contrast to the RME approach presented above, the weights of the formulas have no declarative probabilistic semantics.

Definition 5 (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*. 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 iff their 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, an MLN defines a template for constructing *ground Markov networks*. To each ground Markov network $M_{L,C}$, a probability distribution

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

over possible worlds x is associated, where Z is a normalization factor and $n_i(x)$ compactly expresses the number of true groundings of F_i in the possible world x.

Example 5 *In the following example, we model the relations described in Ex. 1 as an MLN (using the Alchemy syntax (Kok et al. 2008) for MLN files). The "!" operator used in the predicate declarations of* $lives\_in$ *and* $nhood$ *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 express the subjective strength of each rule. We declare the typed predicates* $alarm(person)$*,* $nhood(person, hood\_state!)$*,* $lives\_in(person, town!)$*,* $burglary(person)$*, the types and constants* $person = \{James, Carl\}$*,* $town = \{Yorkshire, Austin\}$*,* $hood\_state = \{Bad, Average, Good\}$*, and add the following weighted formulas:*

$$\begin{aligned}
2.2 \;\; & burglary(x) && => alarm(x) \\
2.2 \;\; & lives_in(x, y) \wedge tornado(y) && => alarm(x) \\
-0.8 \;\; & nhood(x, Good) && => burglary(x) \\
-0.4 \;\; & nhood(x, Average) && => burglary(x) \\
0.4 \;\; & nhood(x, Bad) && => burglary(x)
\end{aligned}$$

Note that, in contrast to BLPs (Ex. 4) and RMEs (Ex. 2) MLNs do not support conditional probabilities (Fisseler 2008), so the rule-like knowledge from Ex. 1 has to be modeled as material implications.

For computing the examples from the previous subsections we employed KREATOR which we will now present in the following section.

3. The KREATOR System

KREATOR 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 designed using the object-oriented programming paradigm. It employs several specialized reasoning tools like Alchemy (Kok et al. 2008) for processing MLNs and SPIRIT (Rödder and Meyer 1996) for (propositional) reasoning with maximum entropy.

System Architecture and Design KREATOR is *modular* and *extensible* with respect to several components. 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. 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 BLPs, MLNs, and RMEs. Other formalisms will be integrated in the near future.

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. As an additional convenient feature KREATOR can export knowledge base files also as formatted LaTeX output.

Working with KREATOR KREATOR comes with a graphical user interface and an integrated console-based interface. The main view of KREATOR is divided into the menu and toolbars and four main panels: the project panel, the editor panel, the outline panel, and the console panel. KREATOR helps the knowledge engineer to organize his work by structuring all data into projects, which may contain e. g. knowledge bases, scripts, and sample/evidence files. The *project panel* of KREATOR gives a complete overview on the project the user is currently working on. The files of a project can be viewed and edited in the *editor panel*, which provides syntax-highlighting and syntax-check for all supported file types. The *outline panel* shows information on the logical components of a knowledge base, such as used predicates, constants, and sorts.

KREATOR features its own scripting language: KREATORSCRIPT incorporates commands for all KREATOR functionalities, so every sequence of working steps can be expressed as an appropriate command sequence in a KREATORSCRIPT file. Every action executed in KREATOR (via GUI or console) is logged as a KREATORSCRIPT command in a *report*. Arbitrary parts of the report can easily be saved as a script file and be executed again when experiments have to be repeated and results have to be reproduced.

Querying Knowledge Bases 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, 4, 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. For each supported knowledge representation formalism, KREATOR internally converts a query in unified syntax to the 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 2 illustrates the processing of a query in unified syntax.

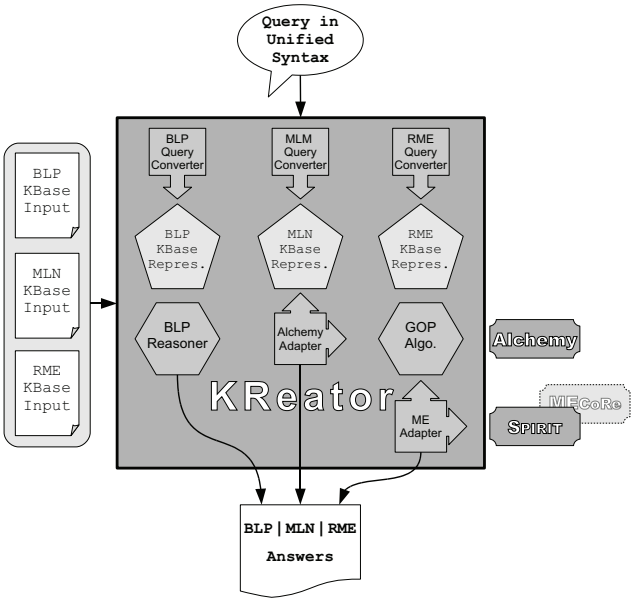

Figure 2: Processing query in unified syntax

4. Comparing RME with Bayes and Markov

Because the knowledge bases in examples 2, 4, and 5 contain just generic knowledge, we add some evidence about the individuals $james$ and $carl$, in order to make differences in knowledge propagation apparent that are caused by having different informations for distinct individuals:

$$lives\_in(james, yorkshire), lives\_in(carl, austin),$$
$$burglary(james), tornado(austin),$$
$$nhood(james) = \mathsf{average}, nhood(carl) = \mathsf{good}$$

The following table shows three queries and their respective probabilities inferred from each of the example knowledge bases and the given evidence:

	BLP	MLN	RME
$alarm(james)$	0.900	0.896	0.918
$alarm(carl)$	0.550	0.900	0.880
$burglary(carl)$	0.300	0.254	0.362

Although each of the three knowledge bases models the same generic knowledge by a different knowledge representation formalism, the inferred probabilities are quite similar, except for the significant lower BLP probability of the query

$alarm(carl)$. This results from using noisy-or as an external combing rule in the BLP calculations while RME and MLN use a more internal information propagation.

Doing further computations in the different formalisms is conveniently supported by KREATOR. For example, dropping $tornado(austin)$ from the evidence yields, as expected, the values for the query $alarm(james)$ as given in the table above; whereas the values for $alarm(carl)$ drop dramatically. Replacing $burglary(james)$ by $alarm(james)$ in the evidence and asking for $burglary(james)$ as in Ex. 1, yields 0.400 (BLP), 0.411 (MLN), and 0.507 (RME).

However, one has to be careful when directly comparing the resulting probabilities, as in fact, each formalism uses different information. Only RME is based on declarative probabilistic conditionals as constraints. The conditional distributions of BLP's carry more information than the MLN and RME knowledge bases, and BLP's are also enriched by conditional independence assumptions. The numbers of MLN formulas are not even probabilities. So, it is more the relative differences of probabilities within each framework that should be compared and interpreted. Again, the BLP shows a result that is most unexpected when comparing the probability of the queries $alarm(james)$ and $alarm(carl)$, whereas MLN and RME behave similar even with respect to relative differences.

5. Summary and Future Work

In this paper we presented the RME approach to relational probabilistic reasoning, applying the principle of maximum entropy to groundings of a relational knowledge base, and introduced the KREATOR system. 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. It already supports Bayesian logic programs, Markov logic networks, and relational maximum entropy.

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 (Fisseler et al. 2007) and other formalisms for relational probabilistic knowledge representation such as and probabilistic relational models (Getoor et al. 2001), as well as to use KREATOR as a testbed to evaluate other approaches for relational probabilistic reasoning under maximum entropy.

While KREATOR already provides a unified query syntax we plan on introducing a unified knowledge base format as well. Having this functionality available will 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://ls1-www.cs.uni-dortmund.de/ie/kreator/`.

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

References

Bacchus, F.; Grove, A.; Halpern, J.; and Koller, D. 1996. From statistical knowledge bases to degrees of belief. *Artificial Intelligence* 87(1-2):75–143.

De Raedt, L., and Kersting, K. 2008. Probabilistic Inductive Logic Programming. In De Raedt, L.; Kersting, K.; Landwehr, N.; Muggleton, S.; and Chen, J., eds., *Probabilistic Inductive Logic Programming*. Springer. 1–27.

Fisseler, J.; Kern-Isberner, G.; Beierle, C.; Koch, A.; and Müller, C. 2007. Algebraic Knowledge Discovery using Haskell. In *Practical Aspects of Declarative Languages, 9th International Symposium*. Springer.

Fisseler, J. 2008. Toward markov logic with conditional probabilities. In Wilson, D. C., and Lane, H. C., eds., *Proceedings of the 21st International FLAIRS Conference*, 643–648. AAAI Press.

Getoor, L., and Taskar, B., eds. 2007. *Introduction to Statistical Relational Learning*. MIT Press.

Getoor, L.; Friedman, N.; Koller, D.; and Tasker, B. 2001. Learning Probabilistic Models of Relational Structure. In Brodley, C. E., and Danyluk, A. P., eds., *Proc. of the 18th International Conf. on Machine Learning (ICML 2001)*. Morgan Kaufmann.

Jaeger, M. 2002. Relational Bayesian Networks: A Survey. *Electronic Transactions in Artificial Intelligence* 6.

Kern-Isberner, G., and Lukasiewicz, T. 2004. Combining probabilistic logic programming with the power of maximum entropy. *Artificial Intelligence, Special Issue on Nonmonotonic Reasoning* 157(1-2):139–202.

Kern-Isberner, G. 1998. Characterizing the principle of minimum cross-entropy within a conditional-logical framework. *Artificial Intelligence* 98:169–208.

Kok, S.; Singla, P.; Richardson, M.; Domingos, P.; Sumner, M.; Poon, H.; Lowd, D.; and Wang, J. 2008. *The Alchemy System for Statistical Relational AI: User Manual*. Department of Computer Science and Engineering, University of Washington.

Loh, S.; Thimm, M.; and Kern-Isberner, G. 2010. Grounding techniques for first-order probabilistic conditional logic. (in preparation).

Muggleton, S., and Chen, J. 2008. A Behavioral Comparison of some Probabilistic Logic Models. In Raedt, L. D.; Kersting, K.; Landwehr, N.; Muggleton, S.; and Chen, J., eds., *Probabilistic Inductive Logic Programming*. Springer. 305–324.

Pearl, J. 1998. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Rödder, W., and Meyer, C.-H. 1996. Coherent Knowledge Processing at Maximum Entropy by SPIRIT. In *Proceedings UAI 1996*, 470–476.

Thimm, M. 2009. Representing Statistical Information and Degrees of Belief in First-Order Probabilistic Conditional Logic. In *Workshop on Relational Approaches to Knowledge Representation and Learning, Proceedings*.