CASAM:

Computer-Aided Semantic Annotation of Multimedia

European Seventh Framework Programme

Project Acronym: **CASAM**

Project Title: **Computer-Aided Semantic Annotation of Multimedia**

Contract Number: **FP7-217061**

Deliverable Number: **D3.2**

Title of the Deliverable: **Basic reasoning engine: Report on optimization techniques for first-order probabilistic reasoning**

Task/WP related to the Deliverable: **WP3**

Type: **Deliverable**

Distribution:

Author(s): **Oliver Gries, Ralf Möller, Anahita Nafissi, Maurice Rosenfeld, Kamil Sokolski, Michael Wessel**

Partner(s) Contributing: **TUHH**

Versioning and Contribution History			
Version	**Date**	**Modification reason**	**Modified by**
01	16/09/09	Reasoning Engine	Anahita Nafissi
02	16/09/09	Probabilisitc Formalism and Evaluation	Oliver Gries
03	25/09/09	Integration, Final Version	Ralf Möller

Executive Summary

The report describes the basic probabilistic reasoning engine for multimedia interpretation (RMI). We introduce the conceptual architecture and explain its basic components. Optimization techniques for implementing specific tasks are identified, and we explain how the architecture is implemented based on modules such as Alchemy and RacerPro. Interfaces are briefly introduced, and the efficiency of the engine as well as the quality of the results is evaluated.

Contents

1 Introduction

Multimedia information retrieval can be improved by relying on high-level symbolic descriptions of media content rather than only on the content itself. High-level content descriptions are called semantic annotations, and the process to acquire these descriptions is called semantic annotation. The project CASAM has the goal to bridge the semantic gap that has hindered the application of fully automated semantic annotation of multimedia content in production environments. The project introduces the concept of computer-aided semantic annotation to accelerate the adoption of semi-automatic multimedia annotations in industry. The main idea is to exploit the synergy of human and machine intelligence to significantly speed up the task of semantic annotation of multimedia content. A human annotator is supported by an automatic component producing initial annotations, which are then revised or completed with a human-computer interaction (HCI) component. The automatic annotation component is composed of an analysis module (KDMA, Knowledge-Driven Multimedia Analysis) and a reasoning-based multimedia interpretation component (RMI). RMI uses a combination of probabilistic and logic-based reasoning.

In this report we describe the RMI conceptual architecture with its basic components. Optimization techniques for realizing specific tasks are identified. We explain how the architecture is implemented in an optimized way based on modules such as Alchemy and RacerPro. Interfaces are briefly introduced, and the efficiency of the engine as well as the quality of the results is evaluated.

In this deliverable we assume familiarity with propositional logic and predicate logic. A basic understanding of probability theory will be helpful.

2 Probabilistic First-Order Knowledge Representation

For representing the domain knowledge used for media interpretation, a combination of probabilistic and logic-based techniques is used. We rely on a first-order logical formalism in order to be able to talk about domain objects and their relations.[1] In principle, predicate logic would be appropriate for this. However, due to knowledge engineering issues we rely on a well-established subset, namely description logics (DLs) [Baader et al., 2003]. This allows us to exploit standard knowledge engineering tools (e.g., Protégé in combination with the reasoner RacerPro) for systematically building knowledge bases with high quality (aka ontologies).

2.1 Preliminaries on Description Logic

One of the main targets of the CASAM project is to support real-time support for annotations. We assume that a less expressive description logic should be applied to facilitate fast computations. We decided to represent the domain knowledge with the DL $\mathcal{ALH}_f{}^-(\mathcal{D})$ (restricted attributive concept language with role hierarchies, functional roles and concrete domains).

In logic-based approaches, atomic representation units have to be specified. The atomic representation units are fixed using a so-called signature. Whereas, up to now, the signature is defined manually, in upcoming deliverables, machine learning techniques might be used to extend the signature automatically such that RMI can be adapted to new contexts.

A DL *signature* is a tuple $\mathcal{S} = (\mathbf{CN}, \mathbf{RN}, \mathbf{AN}, \mathbf{IN})$, where $\mathbf{CN} = \{A_1, ..., A_n\}$ is the set of concept names (denoting sets of domain objects) such as *AssociationActivist*, *Politician*, or *Journalist*. $\mathbf{RN} = \{R_1, ..., R_m\}$ is the set of role names (denoting relations between domain objects). Examples for role names are *interviews* or *hasLocation*. Further, **AN** is a set of concrete domain attributes. Attributes (e.g., *hasValue*, *hasAge*) provide a means to relate domain objects to objects from a so-called concrete domain such as the integers, strings, etc. An introduction to concrete domains in DLs is given in [Baader and Hanschke, 1991]. For brevity, in the following we neglect concrete domains since they are not used in the examples of this report. The signature also contains a component **IN** indicating a set of individuals (names for domain objects).

In order to relate concept names and role names to each other (terminological knowledge) and to talk about specific individuals (assertional knowledge), a knowledge base has to be specified. An $\mathcal{ALH}_f{}^-$ *knowledge base* $\Sigma_\mathcal{S} = (\mathcal{T}, \mathcal{A})$, defined with respect to a signature $\mathcal{S}$, is comprised of a terminological component $\mathcal{T}$ (called *Tbox*) and an assertional component $\mathcal{A}$ (called *Abox*). In the following we just write Σ if the signature is clear from context. A Tbox is a set of so-called *axioms*, which are restricted to the following form in $\mathcal{ALH}_f{}^-$:

(I)	Subsumption	$A_1 \sqsubseteq A_2$, $R_1 \sqsubseteq R_2$
(II)	Disjointness	$A_1 \sqsubseteq \neg A_2$
(III)	Domain and range restrictions for roles	$\exists R.\top \sqsubseteq A$, $\top \sqsubseteq \forall R.A$
(IV)	Functional restriction on roles	$\top \sqsubseteq (\leq 1\, R)$
(V)	Local range restrictions for roles	$A_1 \sqsubseteq \forall R.A_2$
(VI)	Definitions with value restrictions	$A \equiv A_0 \sqcap \forall R_1.A_1 \sqcap ... \sqcap \forall R_n.A_n$

With axioms of form (I), concept (role) names can be declared to be subconcepts (subroles) of each other. Axioms of form (II) denote disjointness between concepts. Axioms of type (III) introduce domain and range restrictions for roles. Axioms of the form (IV) introduce so-called *functional* restrictions on roles, and axioms of type (V) specify local range restrictions (using value restrictions, see below). With axioms of kind (VI) so-called definitions (with necessary and sufficient conditions) can be specified for concept names found on the lefthand side of the $\equiv$ sign. In the axioms, so-called *concepts* are used. Concepts are concept names or expressions of the form $\top$ (anything), $\bot$ (nothing), $\neg A$ (atomic negation), $(\leq 1\, R)$ (role functionality), $\exists R.\top$ (limited existential restriction), $\forall R.A$ (value restriction) and $(C_1 \sqcap ... \sqcap C_n)$ (concept conjunction).

[1] An alternative to first-order logic would be propositional logic. The former is considered as more appropriate in the long run, however.

To illustrate the main ideas we consider now an example Tbox:

$$\mathcal{T} = \{Journalist \sqsubseteq Person, Politician \sqsubseteq \neg Journalist, \exists interviews.\top \sqsubseteq Journalist\}.$$

*Journalist*s are specific *Person*s, and they are disjoint from *Politician*s. The role *interviews* is domain-restricted to *Journalist*, meaning that interviewers are always journalists.

Knowledge about individuals is represented in the Abox part of Σ. An Abox $\mathcal{A}$ is a set of expressions of the form $A(a)$ or $R(a, b)$ (concept assertions and role assertions, respectively) where A stands for a concept name, R stands for a role name, and a, b stand for individuals. Aboxes can also contain equality ($a = b$) and inequality assertions ($a \neq b$). We say that the unique name assumption (UNA) is applied, if $a \neq b$ is added for all pairs of individuals a and b.

We are now ready to consider an example Abox:

$$\mathcal{A} = \{interviews(obj_1, obj_2), Politician(obj_2), Interview(obj_3), hasTopic(obj_3, obj_4), Pollution(obj_4)\}.$$

Due to the Tbox given above it can be concluded, for instance, that obj_1 is a *Journalist* (domain restriction of *interviews*), and obj_1 and obj_2 must denote different domains objects (*Journalist* and *Politician* are disjoint). Specified as decision problems for a knowledge base $\Sigma = (\mathcal{T}, \mathcal{A})$ we determine whether $Journalist(obj_1)$ and $obj_1 \neq obj_2$ is *entailed*. In order to understand the notion of logical entailment , we introduce the semantics of $\mathcal{ALCH}_f{}^-$.

In DLs such as $\mathcal{ALCH}_f{}^-$, the semantics is defined with interpretations $\mathcal{I} = (\triangle^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\triangle^{\mathcal{I}}$ is a non-empty set of domain objects (called the domain of $\mathcal{I}$) and $\cdot^{\mathcal{I}}$ is an interpretation function which maps individuals to objects of the domain ($a^{\mathcal{I}} \in \triangle^{\mathcal{I}}$), atomic concepts to subsets of the domain ($A^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}}$) and roles to subsets of the cartesian product of the domain ($R^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$). The interpretation of arbitrary $\mathcal{ALCH}_f{}^-$ concepts is then defined by extending $\cdot^{\mathcal{I}}$ to all $\mathcal{ALCH}_f{}^-$ concept constructors as follows:

$$\begin{array}{lcl}
\top^{\mathcal{I}} & = & \triangle^{\mathcal{I}} \\
\bot^{\mathcal{I}} & = & \emptyset \\
(\neg A)^{\mathcal{I}} & = & \triangle^{\mathcal{I}} \setminus A^{\mathcal{I}} \\
(\leq 1\, R)^{\mathcal{I}} & = & \{u \in \triangle^{\mathcal{I}} \mid (\forall v_1, v_2)\, [((u, v_1) \in R^{\mathcal{I}} \wedge (u, v_2) \in R^{\mathcal{I}}) \rightarrow v_1 = v_2] \\
(\exists R.\top)^{\mathcal{I}} & = & \{u \in \triangle^{\mathcal{I}} \mid (\exists v)\, [(u, v) \in R^{\mathcal{I}}]\} \\
(\forall R.C)^{\mathcal{I}} & = & \{u \in \triangle^{\mathcal{I}} \mid (\forall v)\, [(u, v) \in R^{\mathcal{I}} \rightarrow v \in C^{\mathcal{I}}]\} \\
(C_1 \sqcap ... \sqcap C_n)^{\mathcal{I}} & = & C_1^{\mathcal{I}} \cap ... \cap C_n^{\mathcal{I}}
\end{array}$$

In the following, the *satisfiability* condition for axioms and assertions of an $\mathcal{ALCH}_f{}^-$-knowledge base Σ in an interpretation $\mathcal{I}$ are defined. A concept inclusion $C \sqsubseteq D$ (concept definition $C \equiv D$) is satisfied in $\mathcal{I}$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ (resp. $C^{\mathcal{I}} = D^{\mathcal{I}}$) and a role inclusion $R \sqsubseteq S$ (role definition $R \equiv S$), if $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ (resp. $R^{\mathcal{I}} = S^{\mathcal{I}}$).

Similarly, assertions $C(a)$ and $R(a, b)$ are satisfied in $\mathcal{I}$, if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ resp. $(a, b)^{\mathcal{I}} \in R^{\mathcal{I}}$. If an interpretation $\mathcal{I}$ satisfies all axioms of $\mathcal{T}$ resp. $\mathcal{A}$ it is called a *model* of $\mathcal{T}$ resp. $\mathcal{A}$. If it satisfies both $\mathcal{T}$ and $\mathcal{A}$ it is called a model of Σ. Finally, if there is a model of Σ (i.e., a model for $\mathcal{T}$ and $\mathcal{A}$), then Σ is called satisfiable.

We are now able to define the entailment relation $\models$. A DL knowledge base Σ *logically entails* an assertion α (symbolically $\Sigma \models \alpha$) if α is satisfied in all models of Σ. For an Abox $\mathcal{A}$, we say $\Sigma \models \mathcal{A}$ if $\Sigma \models \alpha$ for all $\alpha \in \mathcal{A}$.

Referring to the above example, $\Sigma \models Journalist(obj_1)$ and $\Sigma \models obj_1 \neq obj_2$ holds, since $obj_1^{\mathcal{I}} \in Journalist^{\mathcal{I}}$ and $obj_1^{\mathcal{I}} \neq obj_2^{\mathcal{I}}$ in all models $\mathcal{I}$ of Σ.

2.2 Preliminaries on Probabilistic Knowledge Representation

The basic notions for representing probabilistic information are introduced in this section. With the introduction of an assertional language and a query language we adopt a knowledge representation perspective, which slightly deviates from the standard mathematical presentation of probability theory.

2.2.1 Standard Probabilistic Knowledge Representation

The basic notion of probabilistic knowledge representation formalisms is the so-called *random experiment*. For instance, one might pick an arbitrary object from a set of domain objects, and then check whether it is a $Journalist$. The outcome of such a random experiment is denoted by a random variable. A *random variable* X is a function assigning a value to the result of a random experiment. The random experiment itself is not represented, so random variables are functions without arguments, which return different values at different points of time.

In slight misuse of terminology, but in accordance with the literature, the possible values of a random variable comprise the so-called *domain* of the random variable.[2] In the sequel, we will use *boolean* random variables, whose values can be either 1 or 0 ($true$ or $false$, respectively).

Let $\vec{X} = \{X_1, ..., X_n\}$ be the ordered set of all random variables of a (complex) random experiment. An *event* (denoted $\vec{X} = \vec{x}$) is an assignment $X_1 = x_1, ..., X_n = x_n$ to all random variables. In case $n = 1$ we call the event simple, otherwise the event is called complex event. A certain vector of values $\vec{x}$ is referred to as a *possible world*. A possible world can be associated with a *probability value* or *probability* for short. Hence, the notion of a possible world can be used as a synonym for an event, and depending on the context we use the former or the latter name. In case of an event with a boolean random variable X, we write x as an abbreviation for $X = true$ and $\neg x$ as an abbreviation for $X = false$.

A non-deterministic event concerning a random variable X with domain $\{v_1, \ldots, v_n\}$ is denoted as $X = v_1 \vee \ldots \vee X = v_m, m \leq n$. This generalizes to non-deterministic complex events in the obvious way.

Mappings of events to probabilities (or assignment of probabilities to events) are specified with so-called *probability assertions* of the following syntax: $P(\vec{X} = \vec{x}) = p$, where $\vec{X}$ is a vector of random variables, and p is a real value between 0 and 1. An alternative notation also used sometimes is $P(X_1 = x_1, ..., X_n = x_n) = p$. In the special case of a simple event (single random variable, $n = 1$) we write $P(X = x) = p$. The probability value p of an event is denoted as $P(\vec{X} = \vec{x})$ (or $P(X = x)$ in the simple case).

A mapping from the domain of a random variable X to probability values $[0, 1]$ is called a *distribution*. For distributions we use the notation $\mathbf{P}(X)$. Distributions can be defined for (ordered) sets of random variables as well. In this case we use $\mathbf{P}(X_1, \ldots, X_n)$ as a denotation for a mapping to the n-dimensional cross product of $[0, 1]$. For specifying a distribution, probability assertions for *all* domain values must be specified,[3] and the values p must sum up to 1. For an ordered set of random variables $\vec{X}$ with n domain values denoted as $\vec{x}$, a distribution can also be specified in a so-called symbolic form

$$P(\vec{X} = \vec{x}) = f(\vec{x}),$$

where f denotes a function (sometimes called density function, or density for short), which is totally defined on the domain of $\vec{X}$ and has the range $[0, 1]$. This specification is a shorthand for n probability assertions of the form $P(\vec{X} = \vec{x}) = p$ as introduced above.[4] An alternative symbolic notation for a distribution is $\mathbf{P}(\vec{X}) = f$. In its raw form a set of probabilistic assertions, symbolic specifications as a shorthand, is called a *probabilistic knowledge base* (with signature $\vec{X}$).

In case all random variables of a random experiment are mentioned as parameters in a distribution, we speak of a *(full) joint probability distribution* (JPD), otherwise the expression is said to denote a *marginal distribution* (projection of the n-dimensional space of probability values to a lower-dimensional space with m dimensions).

The projection is sometimes combined with a selection: The expression $\mathbf{P}(X_1, \ldots, X_m, X_{m+1} = x_{m+1}, \ldots, X_l = x_l)$ denotes an m-dimensional distribution defined by a selection applied to $\mathbf{P}(X_1, \ldots, X_l)$ such that $X_{m+1} = x_{m+1}, \ldots, X_l = x_l$. After the selection a projection to the dimensions $X_1, \ldots, X_m$ is used. In slight misuse of notation, we sometimes write $\vec{e}$ for a notational fragment $X_i = x_i, \ldots, X_j = x_j$ (e stands for evidence, sometimes another lowercase letter is used). The fragment $X_i = x_i, \ldots, X_j = x_j$ need not necessarily be written at the end in the parameter list of $\mathbf{P}$.

[2] Actually, the set of possible values should be called *range*.

[3] In a sense, knowledge cannot be incomplete in this formalism.

[4] In case the domain is continuous or infinite, without using the symbolic form, the specification of the distribution would not be well-defined.

Let X be a random variable with domain $\{v_1, \ldots, v_n\}$. As usual, in addition to the restriction that probability values are taken from $[0, 1]$ we require that Kolmogorov's axioms are satisfied for the assignments of probabilities to events: $P(X = v_1 \vee \ldots \vee X = v_n) = 1, P(X = v_1 \vee X = v_2) = P(X = v_1) + P(X = v_2) - P(X = v_1, X = v_2)$.

Probabilistic domain knowledge can be specified as a knowledge base, whose semantics is defined to be the full joint probability distribution. For a probabilistic knowledge base KB, formal inference problems (decision or computation problems) are defined.

$$\mathbf{P}(X_1, \ldots, X_n) =?$$

stands for the computation problem of determining the distribution of the random variables $X_1, \ldots, X_n$. The inference problem called *probability query* is denoted as follows.

$$P(X_1 = x_1, \ldots, X_n = x_n) =?$$

stands for the problem to determine the probability value of the event $X_1 = x_1, \ldots, X_n = x_n$ (this can be generalized to non-deterministic events in the obvious way). The entailment decision problem is to check whether adding $P(X_1 = x_1, \ldots, X_n = x_n) = p$ to the knowledge base does not change the joint probability distribution over $X_1, \ldots, X_n$ .This query is particularly useful if the distribution is specified in symbolic form. This, so-called *entailment problem for a probability assertion* is written:

$$KB \models P(X_1 = x_1, \ldots, X_n = x_n) = p.$$

For all these problems, algorithms have been investigated in the literature (see, e.g., [Pearl, 1988]).

2.2.2 Conditional Probabilistic Knowledge Bases

A conditional probability is denoted with $P(\vec{X} = \vec{x} \mid \vec{Y} = \vec{y})$ or, in distribution form, we write $\mathbf{P}(\vec{X} \mid \vec{Y})$ (conditional probability distribution).[5]

$$\mathbf{P}(\vec{X} \mid \vec{Y}) = \frac{\mathbf{P}(\vec{X}, \vec{Y})}{\mathbf{P}(\vec{Y})}, \tag{1}$$

$$\mathbf{P}(\vec{X} \mid \vec{Y}, \vec{z}) = \frac{\mathbf{P}(\vec{X}, \vec{Y}, \vec{z})}{\mathbf{P}(\vec{Y}, \vec{z})}, \tag{2}$$

Related computation problems are conditional probability queries:

$$\mathbf{P}(\vec{X} \mid \vec{Y}) =?$$

$$\mathbf{P}(\vec{X} \mid \vec{Y}, \vec{z}) =?$$

The semantics of these queries are obvious given the definitions of conditional probabilities as defined above. There are various algorithms known in the literature for computing query answers (see [Pearl, 1988]).

A *conditional probabilistic knowledge base* is a probabilistic knowledge base containing also statements of the following form:

$$\mathbf{P}(\vec{X} \mid \vec{Y}, \vec{Z}) = \mathbf{P}(\vec{X} \mid \vec{Y})$$

We call these statements *conditional independence assumptions*. They are extensively used in a classical probabilistic formalism called Bayesian network [Pearl, 1988], which, however, is not considered in this deliverable (see deliverable D3.1 for a discussion about the reasons). Rather than on Bayesian networks, we rely on Markov networks (see also [Pearl, 1988]). In particular we consider a first-order extension called Markov logics [Domingos and Richardson, 2007]. Conditional independence assumptions are specified in a slightly different, but related way in this formalism.

[5] It should be noted that the fraction on the righthand side does not the division of matrices. The whole expression actually denotes a set of equations.

2.2.3 Markov Logic

The formalism of Markov logic [Domingos and Richardson, 2007] provides a means to combine the expressivity of first-order logic augmented with the formalism of Markov networks [Pearl, 1988]. The Markov logic formalism uses first-order logic to define "templates" for constructing Markov networks. The basic notion for this is a called a Markov logic network [Domingos and Richardson, 2007].

A Markov logic network $MLN = (\mathcal{F}, \mathcal{W})$ consists of an ordered multiset of first-order formulas $\mathcal{F} = \{F_1, ..., F_m\}$ and an ordered multiset of real number weights $\mathcal{W} = \{w_1, ..., w_m\}$. The association of a formula to its weight is by position in the ordered sets. For a formula $F \in \mathcal{F}$ with associated weight w we also write wF (weighted formula). Thus, a Markov logic network can also be defined as a set of weighted formulas. Both views can be used interchangeably. As a notational convenience, for ordered sets we nevertheless sometimes write $\vec{X}, \vec{Y}$ instead of $\vec{X} \cup \vec{Y}$.

In contrast to standard first-order logics such as predicate logic, relational structures not satisfying a formula F_i are not ruled out as models. If a relational structure does not satisfy a formula associated with a large weight it is just considered to be quite unlikely the "right" one. For example, the universally quantified formula

$$\forall x \, [RegionWithHighCondensation(x) \rightarrow RegionWithHighPrecipitation(x)]$$

might be true in some relational structures, but might be false in others (there are exceptions). By assigning a reasonable weight to this formula, it becomes a "soft constraint" allowing some relational structures not satisfying this formula to be still considered as possible models (possible worlds). In other words, the relational structure in question corresponds to a world with non-zero probability.

We are now ready to complete the description of the probabilistic first-order representation language. Let $C = \{c_1, ..., c_m\}$ be the set of all constants mentioned in $\mathcal{F}$. A *grounding* of a formula $F_i \in \mathcal{F}$ is a substitution of all variables in the matrix of F_i with constants from C. From all groundings, the (finite) set of grounded atomic formulas (also referred to as *ground atoms*) can be obtained. Grounding corresponds to a domain closure assumption. The motivation is to get rid of the quantifiers and reduce inference problems to the propositional case.

Since a ground atom can either be true or false in an interpretation (or world), it can be considered as a boolean random variable X. Consequently, for each MLN with associated random variables $\vec{X}$, there is a set of possible worlds $\vec{x}$. In this view, sets of ground atoms are sometimes used to denote worlds. In this context, negated ground atoms correspond to *false* and non-negated ones to *true*. We denote worlds using a sequence of (possibly negated) atoms. An example world specification using this convention is:

$$<RegionWithHighCondensation(england), \neg RegionWithHighPrecipitation(england)>$$

When a world $\vec{x}$ violates a weighted formula (does not satisfy the formula) the idea is to ensure that this world is less probable rather than impossible as in predicate logic. Note that weights do not directly correspond to probabilities (see [Domingos and Richardson, 2007] for details).

For each possible world of a Markov logic network $MLN = (\mathcal{F}, \mathcal{W})$ there is a probability for its occurrence. Probabilistic knowledge is required to obtain this value. As usual, probabilistic knowledge is specified using a probability distribution. In the formalism of Markov networks the full joint probability distribution could be specified in symbolic form using a so-called log-linear form (see, e.g., [Domingos and Richardson, 2007]):

$$P(\vec{X} = \vec{x}) = log\_lin(\vec{x}), \tag{3}$$

with log_lin being defined as

$$log\_lin(\vec{x}) = \frac{1}{Z} \, exp \, (\sum_{i=1}^{|\mathcal{F}|} w_i n_i(\vec{x}))$$

According to this definition, the probability of a possible world $\vec{x}$ is determined by the exponential of the sum of the number of true groundings (n_i) of formula $F_i \in \mathcal{F}$ in $\vec{x}$ multiplied with their

corresponding weights w_i and finally normalized with

$$Z = \sum_{\vec{x} \in \vec{X}} exp\,(\sum_{i=1}^{|\mathcal{F}|} w_i n_i(\vec{x})), \tag{4}$$

the sum of the probabilities of all possible worlds. Thus, rather than specifying the full joint distribution directly in symbolic form as we have discussed before, in the Markov logic formalism, the probabilistic knowledge is specified implicitly by the weights associated with formulas. Determining these formulas and their weights in a practical context is all but obvious, such that machine learning techniques are usually employed for knowledge acquisition.

The idea of a Markov logic *network* comes into play because the idea is to assume an edge between two ground atoms (random variables) if they show up in a formula derived by grounding the formulas $\mathcal{F}$. Edges represent influences. Hence, if we assume the grounded formulas derived from $\mathcal{F}$ to be true, atoms co-occurring in a formula somehow influence one another (see the theory about Markov networks [Domingos and Richardson, 2007]). We neglect the network view here because this view is not of much importance for us. What should be noted, however, is the relation of worlds to relational structures as known in first-order logic. A world is given by events concerning random variables corresponding to unary atoms (e.g., $Journalist(oprah)$), two-place atoms (e.g., $interviews(oprah, obama)$), or maybe, in the case of predicate logic, also n-ary atoms. Given truth values for these atoms, we can directly identify the relational structure (in the sense of first-order logic) that corresponds to a world (in sense of probability theory).

For a knowledge base MLN, i.e., a Markov logic network, inference problems can be defined. Let $\vec{X} = \vec{Q} \cup \vec{E} \cup \vec{H}$ be a partition of all random variables $X_1, ..., X_n$, where $\vec{Q}$ is the set of query random variables, $\vec{E}$ is the set of evidence random variables (with known values $\vec{e}$) and $\vec{H}$ is the set of hidden random variables. Given a knowledge base (MLN) involving the random variables $\vec{X}$, a *probability query* is specified as

$$P(\vec{Q} \mid \vec{e}, \vec{H}) =? \tag{5}$$

Exact Inference: Exact algorithms for solving this problem by summing out all hidden random variables provide for a naive way to compute query results. Due to the grounding process, there can be very many hidden variables. Therefore, exact inference is known to be highly intractable even in the case of large numbers of independence assumptions (in MLNs they are implicitly specified by the network structure, but we cannot explain details due to space constraints).

Approximate Inference: Since probability distributions applied to problems in the real world can be very complex, with probabilities varying greatly over a high-dimensional space, there may be no way to sensibly characterize such distributions analytically [Neal, 1993]. Thus, the combinatorial combination of probability values provides for long runtimes in practice. With sampling algorithms it is possible to avoid this kind of problems. Instead of summing out all hidden random variables, the primitive element in any sampling algorithm is the generation of samples from a known probability distribution [Russell and Norvig, 2003]. For example, instead of computing all possible outcomes of a complex experiment with coin throws, the idea is to "flip" the coin itself a number of times.

Markov Chain Monte Carlo (MCMC) algorithms [Gilks et al., 1996] [Neal, 1993] are special cases of sampling algorithms, where the successive sample does only depend on the actual sample and not on its predecessors. Following this idea, with MCMC there is a random walk around the set of possible worlds. It has been proven that the algorithm likely has the tendency to stay in regions of the set of possible worlds that are of higher corresponding probability. Therefore, the more samples are generated, the more appropriate the estimated probabilities will be.

As with exact inference, the efficiency of MCMC does depend on the number of possible worlds. But compared to exact inference, the computation of inference problems with MCMC algorithms can be faster by several orders of magnitude.

For more information to first-order logic and Markov logic and the derivation of conditional independence assumptions we refer to deliverable D3.1.

2.3 Combining Markov Logic and Description Logic

Since $\mathcal{ALH}_f{}^-$ is a fragment of first-order logic, its extension to the Markovian style of formalisms is specified in a similar way as for predicate logic in the section before. The formulas in Markov logic correspond to Tbox axioms and Abox assertions. Weights in Markov description logics are associated with axioms and assertions.

Groundings of Tbox axioms are defined analogously to the previous case.[6] Abox assertions do not contain variables and are already grounded. Note that since an $\mathcal{ALH}_f{}^-(\mathcal{D})$ Abox represents a relational structure of domain objects, it can be directly seen as a possible world itself if assertions not contained in the Abox are assumed to be false.

For appropriately representing domain knowledge in CASAM, weights are possibly used only for a subset of the axioms of the domain ontology. The remaining axioms are assumed to be *strict*, i.e., assumed to be true in any case. A consequence of specifying strict axioms is that lots of possible worlds $\vec{x}$ can be ruled out (i.e., will have probability 0 by definition).

As with Tbox axioms, there is also the need to allow for weighted as well as strict Abox assertions. While the main part of weighted assertions is supposed to be obtained from the multimedia analysis component KDMA, assertions given by the users of the system could be considered to be strict by default, as long as there is nothing known that prevents this.

A *Markov DL knowledge base* Σ_M is a tuple $(\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is comprised of a set $\mathcal{T}_s$ of strict axioms and a set $\mathcal{T}_w$ of weighted axioms and $\mathcal{A}$ is comprised of a set $\mathcal{A}_s$ of strict assertions and a set $\mathcal{A}_w$ of weighted assertions.

Referring to axioms, a proposal for CASAM is to consider strictness for the domain ontology patterns (I)–(IV):[7]

(I)	subsumption	$A_1 \sqsubseteq A_2,\ R_1 \sqsubseteq R_2$
(II)	disjointness	$A_1 \sqsubseteq \neg A_2$
(III)	domain and range restrictions	$\exists R.\top \sqsubseteq A,\ \top \sqsubseteq \forall R.A$
(IV)	functional roles	$\top \sqsubseteq (\leq 1\, R)$

The main justification treating axioms as strict is that the subsumption axioms, disjointness axioms, domain and range restrictions as well as functional role axioms (in combination with UNA) are intended to be true in any case such that there is no need to assign large weights to them. Weights are often assigned to axioms representing definitions (see above) as well as special assertions for information obtained from multimedia analysis (e.g. equality axioms).

The advantage of this probabilistic approach is that initial ontology engineering is done as usual with standard reasoning support and with the possibility to add weighted axioms and weighted assertions on top of the strict fundament. Since lots of possible worlds do not have to be considered because their probability is known to be 0, probabilistic reasoning will be significantly faster. We discuss these issues with an example. In order to keep the example simple, it does not consider binary atoms (but the general structure would be the same).

Consider a Markov DL KB with $\mathcal{T}_s = \{Car \sqsubseteq Vehicle, Vehicle \sqsubseteq \neg Forest\}$, $\mathcal{A}_w = \{1.1\ Car(ind_1), 0.6\ Forest(ind_1)\}$ and $\mathcal{T}_w$ as well as $\mathcal{A}_s$ being empty. There are 2^3 ground atoms, but due to the strict axioms there are only four possible worlds:

$$\vec{x_1} = <Car(ind_1), Vehicle(ind_1), \neg Forest(ind_1)>$$
$$\vec{x_2} = <\neg Car(ind_1), \neg Vehicle(ind_1), Forest(ind_1)>$$
$$\vec{x_3} = <\neg Car(ind_1), Vehicle(ind_1), \neg Forest(ind_1)>$$
$$\vec{x_4} = <\neg Car(ind_1), \neg Vehicle(ind_1), \neg Forest(ind_1)>$$

The full joint probability distribution is specified by the elements of $\mathcal{A}_w$. The following probabilistic assertions are entailed:[8] $P(\vec{X} = \vec{x_1}) = \frac{exp(1.1)}{Z} = \frac{3.004166}{Z}$, $P(\vec{X} = \vec{x_2}) = \frac{exp(0.6)}{Z} = \frac{1.8221188}{Z}$ and $P(\vec{X} = \vec{x_i}) = \frac{exp(0)}{Z} = \frac{1}{Z}$, $i = 3, 4$, $Z = 6.8262848$ such that e.g. $P(\vec{X} = \vec{x_1}) \approx 0.44$.

It is instructive to consider the relation of weights as used in Markov logic and probabilities assigned to events. As we have seen above, Abox assertions are ground formulas by definition.

[6] For this purpose, the variable-free syntax of axioms can be first translated to predicate logic.

[7] But there may be exceptions.

[8] We took the freedom to refine the value on the right hand side in the standard mathematical style, but the reader should be aware of the fact that we talk about propositional assertions that are entailed by the knowledge base.

Thus, Abox assertions correspond to (boolean) random variables in the Markov logic framework. Let us consider events for such a boolean random variable about which we have no information. Hence, the probability of any event should be 0.5. It seems to be obvious that in this case the weight of the assertion is 0. By referring to the symbolic form of the joint probability distribution this can be confirmed: Let $\mathcal{A} = \{0.0\ A(ind_1)\}$ be a weighted Abox. Then $P(A(ind_1)) = \frac{e^0}{e^0+e^0} = \frac{1}{2}$ is entailed. The probability of arbitrary weighted atomic concept assertions $w\ A(ind_i)$ is determined as follows:

$$p = \frac{e^w}{e^w + e^0}, \tag{6}$$

In other words $P(A(ind_i)) = p$ is entailed. The term is correct for weighted role assertions $w\ R(ind_i, ind_j)$ as well. If $w = -\infty$, as expected $p = 0$, and if $w = \infty$, as expected $p = 1$. In order to determine the weight for probabilistic assertions given the probability of the associated event, (6) has to be resolved to w and the result is

$$w = ln\left(\frac{p}{1-p}\right). \tag{7}$$

It should be noted that the weight computed this way might have to be adapted if the assertion is considered in the context of a complete knowledge base. All axioms and assertions contribute to the full joint distribution.

Assertions given by the knowledge-driven multimedia analysis component (KDMA) and the human computer interaction component (HCI) – if not strict – are expected to be positive, i.e., to have a probability greater than 0.5. While probabilities lower than 0.5 indicate that the assertion given is believed to be rather false than true (an information that is not appropriate for CASAM), probabilities equal to 0.5 indicate that the truth of the assertion is completely unknown (providing no new information). In case degrees of beliefs lower than 0.5 are intended to be positive ("The assertion is true with degree of belief 0.1, but this does not mean that the opposite is true with degree of belief 0.9"), RMI will transform degrees of beliefs d to probabilities p with

$$p = 0.5 + (d/2). \tag{8}$$

If, e.g., the positively intended degree of belief is 0.1, $p = 0.5 + (0.1/2) = 0.55$. Independent of their interpretation, probabilities should be assigned with minimal arbitrariness. RMI is only able to provide reasonable results if the probabilities of the assertions are reasonable. Therefore, it is important that components providing assertions, especially KDMA, will give justifications of the probabilities of their assertions.

3 Reasoning-based Media Interpretation: RMI – The Basic Engine

In this chapter the basic interpretation engine RMI is introduced. The architecture will be explained at the conceptual level in the first section, optimization techniques will be discussed afterwards in the implementation section. Interfaces to RMI and its subcomponents are presented in order to show how the architecture is used in the whole CASAM framework. An initial evaluation concludes the description of RMI.

3.1 RMI Conceptual Architecture

Figure 1 depicts the basic components at the conceptual level with input and output data (Aboxes in both cases), intermediate results (Aboxes) as well as background knowledge (a Tbox and a set of rules) used by different modules.[9]

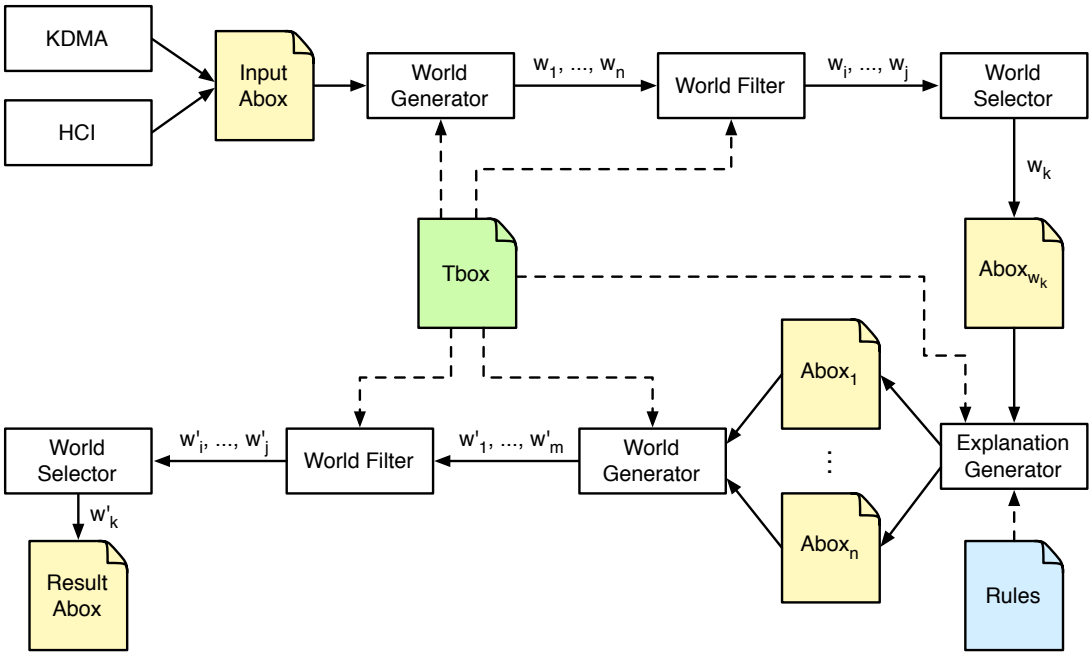

Figure 1: Conceptual view of the reasoning-based media interpretation engine.

The signature of the knowledge base contains appropriate names required for representing knowledge of the domain we use in CASAM, namely environmental issues and political interviews, as well as names required for representing information about document structures (MCO, media content ontology). Appropriate axioms to relate these names to one another are found in the Tbox.

From the input Abox all assertions involving names from MCO are removed for computing interpretations. These assertions describing the document structure are not probabilistic, and they slow down the inference process but do not contribute to the result Abox. The simple preprocessing step is not shown in Figure 1. The preprocessed Abox (Input Abox) is then further processed using three subsequent units.

- World Generator: Based on the Tbox and the preprocessed input Abox this component produces all Markov logic worlds, which are indicated in Figure 1 by $w_1, w_2, \ldots, w_n$. As we have discusses before, a world is a vector of ground atoms. Assume there are m ground atoms. Consequently, the number of worlds is 2^m. The generated worlds are the input to the next component called World Filter.

[9]The background knowledge could also contain Abox assertions to represent background knowledge about individuals. We neglect this here, however, in order to simplify the presentation.

- World Filter: This component removes the impossible worlds. In this step, the subsumption axioms and domain and range restrictions in the Tbox are considered for world elimination. The remaining worlds $w_i, \ldots, w_j$, also known as possible worlds, are the input to the next component called World Selector.

- World Selector: This component selects the most-probable world among the set of possible worlds. The most-probable world has the highest probability based on Markov logic as explained above. The most-probable world w_k is transformed into an Abox (called Abox_{w_k}).

The Abox_{w_k} is then used as input for the Explanation Generator. The idea is to generate further evidence for the percepts obtained as input by using domain knowledge to generate new assertions from which the assertions representing the percepts can be derived. Multiple explanations are possible. Therefore, multiple output Aboxes can occur (see Figure 1). For each of those explanations the most-probable world is computed using the three steps World Generator, World Filter, and World Selector that are described above. There might be several worlds with the same maximal probability, thus, in principle, there might be multiple output Aboxes. In the basic RMI engine, one of them is chosen non-deterministically, however (see Chapter 4).

Note again that this is a conceptual view on the architecture. In the following we show how the architecture is implemented in an optimized way. The units dealing with worlds are implemented using the Alchemy system, and the explanation facility is realized with a module called CASAM Abduction Engine (CAE). Actually, Alchemy is used to approximate the functionality of the three units in order to better satisfy real-time requirements. Figure 2 summarizes the implementation-based view of RMI using Alchemy and CAE (which is based on RacerPro). Alchemy is applied to each of the Aboxes $A_1 to A_n$ (the diagram is slightly simplified).

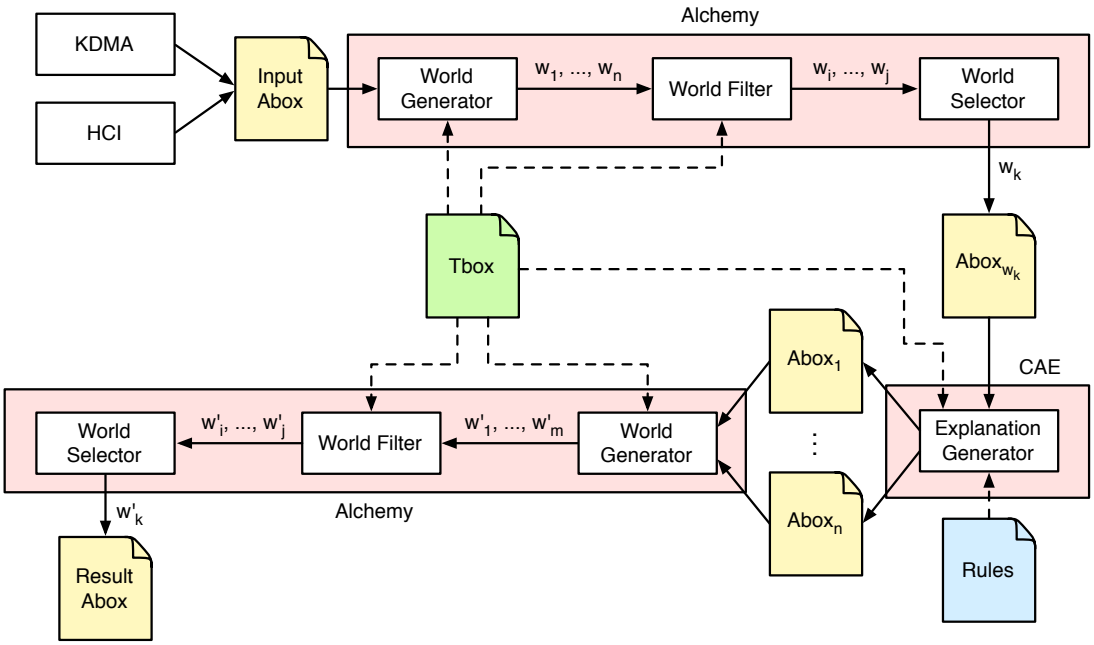

Figure 2: Implementation-oriented view of the RMI engine.

3.2 Markov Logic Engine: Alchemy

Alchemy is an open-source software package developed at the University of Washington. Alchemy supports Markov logics as the underlying formalism. The system provides algorithms for probabilistic logic inference and statistical relational learning.

3.2.1 Alchemy Knowledge Representation Language

Knowledge in Alchemy is specified as weighted first-order logic formulas, the formulas are built with the following standard predicate logic operators.[10].

- In Alchemy different names refer to different objects (UNA)
- Alchemy applies the domain closure assumption: Existentially quantified formulas are replaced by disjunctions of groundings of the quantified formula with constants used in the knowledge base. Universal quantifiers are replaced by corresponding groundings as well.

An example for the domain closure assumption is helpful to clarify the consequences. Assume the following formula $F = \exists x, y\,[MotherOf(x, y)]$ and two constants $Mary$ and $Sara$. Applying the domain closure assumption means that the formula F can be replaced with $MotherOf(Mary, Mary)$ $\vee MotherOf(Mary, Sara) \vee MotherOf(Sara, Mary) \vee MotherOf(Sara, Sara)$.

For answering probability queries, the number of true groundings need to be determined (see Section 2.2.3). Thus, answering probability queries corresponds to model-checking first-order formulas. The problem of answering probability queries is known to be PSPACE-complete. The same holds for the entailment problem for a probability assertion.

A knowledge base consists of three parts namely *types*, *predicates* and *formulas*. The first two parts are required whereas the last part is optional.

1. Types: In the first part, types are defined and a set of constants is assigned to each defined type e.g. $city = \{Hamburg, Berlin\}$ indicates a type *city* with two constants *Hamburg* and *Berlin*. Each defined type must have at least one constant. In Alchemy, a constant can have different types. The advantage of using types is that the inference process speeds up since the world generator of Alchemy produces only the worlds which correspond to correct typings.

2. Predicates: In the second part, used predicates and their applied types are introduced e.g. *AirPollution(city)* defines a predicate *AirPollution* with type *city*.

3. Formulas: In the last part, hard- and soft formulas (in first-order logic) are listed. Hard formulas, i.e., strict formulas, are terminated by a period in Alchemy, and soft formulas are preceded by a weight. Alchemy assigns hard formulas internally a high weight. Worlds violating the hard formulas are not impossible since Alchemy assigns them a negligible probability. Consequently, these worlds are not removed by the world filter component of the basic reasoning engine. Assume two predicates *Industry* and *AirPollution* which are of type *city*. In the following, an example for a hard formula is given:

$$Industry(x) \Rightarrow AirPollution(x). \tag{9}$$

   Although there are possible worlds not satisfying (9). Due to the typing constraints, the variable x can only be substituted with individuals of type *city*. Similarly, consider two predicates *Rain* and *Flood* of type *city*. The next example shows a soft formula:

$$0.1\ \ Rain(x) \Rightarrow Flood(x) \tag{10}$$

   This means that rain leads to flood with a weight of *0.1*. Note that weighted assertions are also written in the formulas part of the MLN file . The weights of formulas are either *hand-crafted* or *learned*. Learning weights can be performed by the weight learning tool of Alchemy.

3.2.2 Inference Services

Alchemy can solve the problem of answering probability queries as well as the entailment problem for probability assertions using exact inference as described above. Runtimes in practice might turn out to be too long, however. Therefore, Alchemy can be instructed to perform approximate

[10]! is used to denote negation

inference. The exactness of approximate inference can be controlled. In order to produce more accurate results, the maximum number of steps to run sampling algorithms can be increased (option -maxSteps). By increasing the number of samples, the results of approximate inference converge to the results of exact inference.

Some effects of the approximation techniques used by Alchemy have to be understood, however. We have seen that strict formulas can reduce the number of worlds that have non-zero probability. Based on the theory of Markov logics, the weight of a strict formula is positive infinity. According to the manual, Alchemy assigns (positive) finite weights to strict formulas, however. To determine these weights for strict formulas, Alchemy converts input formulas into conjunctive normal form (CNF). Afterwards, the weight of a formula is divided equally among its CNF clauses.

The weight assigned to a strict formula depends on the inference type. Alchemy performs two types of inference, namely:

- Probability queries: Probabilistic inference methods currently implemented in Alchemy are based on two general algorithms namely, Markov Chain Monte Carlo (MCMC) and (lifted) belief propagation (option -bp). Based on Markov Chain Monte Carlo (MCMC) different inference algorithms have been implemented in Alchemy namely, Gibbs sampling (option -p), simulated tempering (option -simtp), and MC-SAT (option -ms). Note that the default algorithm is lifted belief propagation. The advantage of lifted inference in comparison to the fully grounded network is the runtime and memory usage. For the above inference methods, the number of algorithm iterations can be specified. The default weight assigned to the clauses of a strict formula based on MCMC inference is twice the maximum weight mentioned in the MLN. The output file of a probabilistic inference consists of probabilities that the query atoms are true.

- MAP inference: This type of inference is called Maximum A Posteriori (MAP) (option -a) and returns the most-likely state of query atoms given the evidence. In other words, the output file consists of atoms associated with zeros and ones (denoting a world). The default weight assigned to the clauses of a strict formula based on MAP inference is the sum of appearing in MLN plus 10.

RMI applies MAP inference, and therefore this algorithm is discussed in more detail now. Based on MAP inference, the most probable world given the evidence is determined. As it was mentioned before, the world filter removes the impossible words. Afterwards, the world selector based on MAP determines which world among the possible worlds has the highest probability. To determine it, the argument y of the conditional probability $P(\vec{x} \mid \vec{y})$ is maximized:

$$\arg\max_{\vec{x}} \; P(\vec{x} \mid \vec{y}) \tag{11}$$

where $\vec{y}$ indicates query predicate(s) and $\vec{x}$ evidence(s). By replacing the above conditional probability with the Markovian formula, it follows:

$$\arg\max_{\vec{x}} \; \frac{1}{Z_x} \exp\left(\sum_i w_i n_i \left(\vec{x}, \vec{y}\right)\right) \tag{12}$$

By applying MAP, normalization is done for worlds corresponding to the given evidence. Keeping in mind that the exponential function is monotonic, only the argument of the exponential function has to be maximized:

$$\arg\max_{\vec{x}} \; \sum_i w_i n_i \left(\vec{x}, \vec{y}\right) \tag{13}$$

3.2.3 Interfaces to Alchemy

The command which performs inference in Alchemy is *infer* and it has the following form:

```
infer -i uniform.mln -r uniform.result -e empty.db -q QueryFormula
```

The options indicate:

- -i: input file “uniform.mln”
- -r: output file “uniform.result”
- -e: evidence file “empty.db”
- -q: query formula

In the following, the above mentioned file types for performing inference are introduced:

- Markov logic network file (with extension .mln, required): The MLN file contains types, predicates, and formulas as introduced above.
- Evidence file (with extension .db, optional): The DB file contains evidence ground atoms which can be either true or false. By default, evidence predicates are treated using the closed-world assumption, meaning that if they are not present in the DB file, they are assumed to be false. Non-evidence predicates are treated using the open-world assumption by default (a predicate is called an evidence predicate if at least one grounding of a predicate exists in the evidence file.). The evidence file can be empty, since it is optional. It is also possible to use multiple evidence files.
- Output file (with extension .result): In case of performing probabilistic inference, this file contains the probabilities of query atoms given the evidence file. In case of MAP inference, this file shows the most likely state of query atoms.

In this section, an example for a MLN file is given.

$$
\begin{array}{l}
city = \{Hamburg, Berlin\} \\
Industry\,(city) \\
AirPollution\,(city) \\
Rain\,(city) \\
Flood\,(city) \\
Industry(x) \Rightarrow AirPollution(x). \\
0.1\ Rain(x) \Rightarrow Flood(x) \\
0.3\ Industry\,(Hamburg)
\end{array}
$$

Table 1: Example for a MLN file

Additionally, an example for a DB file based on the above MLN file is given:

$$
\begin{array}{l}
Rain\,(Hamburg) \\
AirPollution\,(Berlin) \\
Flood\,(Berlin)
\end{array}
$$

Table 2: Example for a DB file

A query formula for this example could be:

$$AirPollution(Hamburg) \wedge Flood(Hamburg) \tag{14}$$

3.3 CASAM Abduction Engine Implemented with RacerPro

The RMI engine has a component called CASAM Abduction Engine (CAE) which is responsible for generating so-called explanations for the assertions found in the input Aboxes. The idea of computing explanations is to derive “additional support” for Abox assertion which should be

satisfied in all models. In our input context, the input Abox describes observations (generated by KDMA and HCI). CAE uses the RacerPro abduction inference service to compute explanations for (single) Abox assertions. We start with an introduction to preliminaries before details of the CASAM Abduction Engine are explained.

3.3.1 Sequences, Variable Substitutions and Transformations

For the introduction of the abduction algorithm, we need some additional definitions. A *variable* is a name of the form ?*name* where *name* is a string of characters from $\{a..z\}$. In the following definitions, we denote places where variables can appear with uppercase letters.

Let V be a set of variables, and let $\underline{X}, \underline{Y_1}, \ldots, \underline{Y_n}$ be sequences $\langle \ldots \rangle$ of variables from V. $\underline{z}$ denotes a sequence of individuals. We consider sequences of length 1 or 2 only, if not indicated otherwise, and assume that $(\langle X \rangle)$ is to be read as (X) and $(\langle X, Y \rangle)$ is to be read as (X, Y) etc. Furthermore, we assume that sequences are automatically flattened. A function *as_set* turns a sequence into a set in the obvious way.

A *variable substitution* $\sigma = [X \leftarrow i, Y \leftarrow j, \ldots]$ is a mapping from variables to individuals. The application of a variable substitution σ to a sequence of variables $\langle X \rangle$ or $\langle X, Y \rangle$ is defined as $\langle \sigma(X) \rangle$ or $\langle \sigma(X), \sigma(Y) \rangle$, respectively, with $\sigma(X) = i$ and $\sigma(Y) = j$. In this case, a sequence of individuals is defined. If a substitution is applied to a variable X for which there exists no mapping $X \leftarrow k$ in σ then the result is undefined. A variable for which all required mappings are defined is called *admissible* (w.r.t. the context).

3.3.2 Grounded Conjunctive Queries

Let $\underline{X}, \underline{Y_1}, \ldots, \underline{Y_n}$ be sequences of variables, and let $Q_1, \ldots, Q_n$ denote concept or role names.

A query is defined by the following syntax.

$$\{(\underline{X}) \mid Q_1(\underline{Y_1}), \ldots, Q_n(\underline{Y_n})\}$$

The sequence $\underline{X}$ may be of arbitrary length but all variables mentioned in $\underline{X}$ must also appear in at least one of the $\underline{Y_1}, \cdots, \underline{Y_n}$: $as\_set(\underline{X}) \subseteq as\_set(\underline{Y_1}) \cup \cdots \cup as\_set(\underline{Y_n})$.

Informally speaking, $Q_1(\underline{Y_1}), \ldots, Q_n(\underline{Y_n})$ defines a conjunction of so-called *query atoms* $Q_i(\underline{Y_i})$. The list of variables to the left of the sign $\mid$ is called the *head* and the atoms to the right of are called the query *body*. The variables in the head are called distinguished variables. They define the query result. The variables that appear only in the body are called non-distinguished variables and are existentially quantified.

Answering a query with respect to a knowledge base Σ means finding admissible variable substitutions σ such that $\Sigma \models \{\sigma(Q_1(\underline{Y_1})), \ldots, \sigma(Q_n(\underline{Y_n}))\}$. We say that a variable substitution $\sigma = [X \leftarrow i, Y \leftarrow j, \ldots]$ introduces *bindings* $i, j, \ldots$ for variables $X, Y, \ldots$. Given all possible variable substitutions σ, the *result* of a query is defined as $\{(\sigma(\underline{X}))\}$ Note that the variable substitution σ is applied before checking whether $\Sigma \models \{Q_1(\sigma(\underline{Y_1})), \ldots, Q_n(\sigma(\underline{Y_n}))\}$, i.e., the query is *grounded* first.

For a query $\{(?y) \mid Person(?x), hasParticipant(?y, ?x)\}$ and the Abox $\Gamma_1 = \{HighJump(ind_1)$, $Person(ind_2)$, $hasParticipant(ind_1,\ ind_2)\}$, the substitution $[?x \leftarrow ind_2, ?y \leftarrow ind_1]$ allows for answering the query, and defines bindings for $?y$ and $?x$.

A *boolean* query is a query with $\underline{X}$ being of length zero. If for a boolean query there exists a variable substitution σ such that $\Sigma \models \{\sigma(Q_1(\underline{Y_1})), \ldots, \sigma(Q_n(\underline{Y_n}))\}$ holds, we say that the query is answered with *true*, otherwise the answer is *false*.

Later on, we will have to convert query atoms into Abox assertions. This is done with the function *transform*. The function *transform* applied to a set of query atoms $\{\gamma_1, \ldots \gamma_n\}$ is defined as $\{transform(\gamma_1, \sigma), \ldots, transform(\gamma_n, \sigma)\}$ where
$transform(P(\underline{X}), \sigma) := P(\sigma(\underline{X}))$.

3.3.3 Rules

A rule r has the following form $P(\underline{X}) \leftarrow Q_1(\underline{Y_1}), \ldots, Q_n(\underline{Y_n})$ where P, $Q_1, \ldots, Q_n$ denote concept or role names with the additional restriction (safety condition) that $as\_set(\underline{X}) \subseteq as\_set(\underline{Y_1}) \cup \cdots \cup as\_set(\underline{Y_n})$. For instance, for *AirPollution* the following rules might be used.

$$AirPollution(X) \leftarrow City(X), Industry(Y), near(X,Y).$$
$$AirPollution(X) \leftarrow TrafficJam(X).$$

Rules are used to derive new Abox assertions, and we say that a rule r is *applied* to an Abox $\mathcal{A}$. The function call $apply(\Sigma, P(\underline{X}) \leftarrow Q_1(\underline{Y_1}), \ldots, Q_n(\underline{Y_n}), \mathcal{A})$ returns a set of Abox assertions $\{\sigma(P(\underline{X}))\}$ if there exists an admissible variable substitution σ such that the answer to the query

$$\{() \mid Q_1(\underline{Y_1}), \ldots, Q_n(\underline{Y_n})\}$$

is *true* with respect to $\Sigma \cup \mathcal{A}$.[11] If no such σ can be found, the result of the call to $apply(\Sigma, r, \mathcal{A})$ is the empty set. The application of a set of rules $\mathcal{R} = \{r_1, \ldots r_n\}$ to an Abox is defined as follows.

$$apply(\Sigma, \mathcal{R}, \mathcal{A}) = \bigcup_{r \in \mathcal{R}} apply(\Sigma, r, \mathcal{A})$$

The result of $forward\_chain(\Sigma, \mathcal{R}, \mathcal{A})$ is defined to be $\emptyset$ if $apply(\Sigma, \mathcal{R}, \mathcal{A}) \cup \mathcal{A} = \mathcal{A}$ holds. Otherwise the result of *forward_chain* is determined by the recursive call
$apply(\Sigma, \mathcal{R}, \mathcal{A}) \cup forward\_chain(\Sigma, \mathcal{R}, \mathcal{A} \cup apply(\Sigma, \mathcal{R}, \mathcal{A}))$.

For some set of rules $\mathcal{R}$ we extend the entailment relation by specifying that $(\mathcal{T}, \mathcal{A}) \models_{\mathcal{R}} \mathcal{A}_0$ iff $(\mathcal{T}, \mathcal{A} \cup forward\_chain((\mathcal{T}, \emptyset), \mathcal{R}, \mathcal{A})) \models \mathcal{A}_0$.[12]

3.3.4 Computing Explanations via Abduction

In general, abduction is formalized as $\Sigma \cup \Delta \models_{\mathcal{R}} \Gamma$ where background knowledge (Σ), rules ($\mathcal{R}$), and observations (Γ) are given, and explanations (Δ) are to be computed. In terms of DLs, Δ and Γ are Aboxes and Σ is a pair of Tbox and Abox.

Abox abduction is implemented as a non-standard retrieval inference service in DLs. In contrast to standard retrieval inference services where answers are found by exploiting the ontology, Abox abduction has the task of acquiring what should be added to the knowledge base in order to answer a query. Therefore, the result of Abox abduction is a set of hypothesized Abox assertions. To achieve this, the space of abducibles has to be previously defined and we do this in terms of rules.

We assume that a set of rules $\mathcal{R}$ as defined above (see Section 3.3.3) are specified, and define a non-deterministic function *compute_explanation* as follows.

- $compute\_explanation(\Sigma, \mathcal{R}, \mathcal{A}, P(\underline{Z})) = transform(\Phi, \sigma)$ if there exists a rule $r = P(\underline{X}) \leftarrow Q_1(\underline{Y_1}), \ldots, Q_n(\underline{Y_n}) \in \mathcal{R}$ that is applied to an Abox $\mathcal{A}$ such that a minimal set of query atoms Φ and an admissible variable substitution σ with $\sigma(\underline{X}) = \underline{Z}$ can be found, and the query $Q := \{() \mid expand(P(\underline{X}), r, \mathcal{R}) \setminus \Phi\}$ is answered with *true*.
- If no such rule r exists in $\mathcal{R}$ it holds that $compute\_explanation(\Sigma, \mathcal{R}, \mathcal{A}, P(\underline{Z})) = \emptyset$.

The goal of the function *compute_explanation* is to determine what must be added (Φ) such that an entailment $\Sigma \cup \mathcal{A} \cup \Phi \models_{\mathcal{R}} P(\underline{Z})$ holds. Hence, for *compute_explanation*, abductive reasoning is used. The set of query atoms Φ defines what must be hypothesized in order to answer the query Q with *true* such that $\Phi \subseteq expand(P(\underline{X}), r, \mathcal{R})$ holds. The definition of *compute_explanation* is non-deterministic due to several possible choices for Φ.

The function application $expand(P(\underline{X}), P(\underline{X}) \leftarrow Q_1(\underline{Y_1}), \ldots, Q_n(\underline{Y_n}), \mathcal{R})$ is also defined in a non-deterministic way as

$$expand'(Q_1(\underline{Y_1}), \mathcal{R}) \cup \cdots \cup expand'(Q_n(\underline{Y_n}), \mathcal{R})$$

with $expand'(P(\underline{X}), \mathcal{R})$ being $expand(P(\underline{X}), r, \mathcal{R})$ if there exist a rule $r = P(\underline{X}) \leftarrow \ldots \in \mathcal{R}$ and $\langle P(\underline{X}) \rangle$ otherwise. We say the set of rules is backward-chained, and since there might be multiple rules in $\mathcal{R}$, backward-chaining is non-deterministic as well. Thus, multiple explanations are generated.[13]

[11] We slightly misuse notation in assuming $(\mathcal{T}, \mathcal{A}) \cup \Delta = (\mathcal{T}, \mathcal{A} \cup \Delta)$. If $\Sigma \cup \mathcal{A}$ is inconsistent the result is well-defined but useless. It will not be used afterwards.

[12] We could also give a semantic definition of entailment w.r.t. a set of rules without using *forward_chain*. However, in this deliverable we do not attempt to prove that the abduction algorithm is correct. Thus, only proof-theoretic definition is given.

[13] In the expansion process, variables have to be renamed. We neglect these issues here.

3.3.5 The Main Procedure

In the following we devise an abstract computational engine for "explaining" Abox assertions in terms of a given set of rules. Explanation of Abox assertions w.r.t. a set of rules is meant in the sense that using the rules some high-level explanation is constructed such that the Abox assertions are entailed. The explanation of an Abox is again an Abox. For instance, the output Abox represents results of the RMI content interpretation process. The presentation in slightly extended compared to the one in [Castano et al., 2008].

Let Γ be an Abox of observations whose assertions are to be explained. The goal of the explanation process is to use a set of rules $\mathcal{R}$ to derive "explanations" for elements in Γ. The explanation algorithm implemented in the CASAM abduction engine works on a set of Aboxes $\mathfrak{I}$.

Initially, $\mathfrak{I} \Leftarrow \{\Gamma\}$, e.g. $\{personNameToCountry(pName_1,\ country_1)$, $sportsNameToCity(hjName_1,\ city_1)\}$, at this stage, the explanation is just the input Abox Γ.[14] The complete explanation process is implemented by the CAE function:

function $CAE(\Omega, \Xi, \Sigma, \mathcal{R}, S, \Gamma)$:
  $\mathfrak{I}' := \{\Gamma\}$
  repeat
    $\mathfrak{I} := \mathfrak{I}'$
    $(\mathcal{A}, \alpha) := \Omega(\mathfrak{I})$ // $\mathcal{A} \in \mathfrak{I}$, $\alpha \in \mathcal{A}$ s.th. $requires\_fiat(\alpha)$ holds
    $\mathfrak{I}' := (\mathfrak{I} \setminus \{\mathcal{A}\}) \cup maximize(\Sigma, \mathcal{R}, \mathcal{A}, explanation\_step(\Sigma, \mathcal{R}, S, \mathcal{A}, \alpha), S)$.
  until $\Xi(\mathfrak{I})$ or no $\mathcal{A}$ and α can be selected such that $\mathfrak{I}' \neq \mathfrak{I}$
  return $\mathfrak{I}$

It takes as parameters a strategy function Ω, a termination function Ξ, a background knowledge Σ, a set of rules $\mathcal{R}$, a scoring function S and an Abox Γ of observations. It applies the strategy function Ω in order to decide which assertion to explain, uses a termination function Ξ in order to check whether to terminate due to resource constraints and a scoring function S to evaluate an explanation.

The function Ω for the explanation strategy and Ξ for the termination condition are used as an oracle and must be defined in an application-specific way. In our multimedia interpretation scenario we assume that the function $requires\_fiat$ is defined in an application-specific way such that it returns true for those assertions which should be explained (usually, it always returns true). The function $explanation\_step$ is defined as follows.

$explanation\_step(\Sigma, \mathcal{R}, S, \mathcal{A}, \alpha)$:

$$\bigcup_{\Delta \in compute\_all\_explanations(\Sigma, \mathcal{R}, S, \mathcal{A}, \alpha)} consistent\_completed\_explanations(\Sigma, \mathcal{R}, \mathcal{A}, \Delta).$$

We need two additional auxiliary functions.

$consistent\_completed\_explanations(\Sigma, \mathcal{R}, \mathcal{A}, \Delta)$:

$$\{\Delta' \mid \Delta' = \Delta \cup \mathcal{A} \cup forward\_chain(\Sigma, \mathcal{R}, \Delta \cup \mathcal{A}), consistent_{\Sigma}(\Delta')\}$$

$compute\_all\_explanations(\Sigma, \mathcal{R}, S, \mathcal{A}, \alpha)$:

$$\{\Delta \mid \Delta = compute\_explanation(\Sigma, \mathcal{R}, \alpha)\}.$$

Note the call to the nondetermistic function $compute\_explanation$. It may return different values, all of which are collected. The function $consistent_{(\mathcal{T}, \mathcal{A})}(\mathcal{A}')$ determines if the Abox $\mathcal{A} \cup \mathcal{A}'$ has a model which is also a model of the Tbox $\mathcal{T}$.

Depending on the application context, some of the observations can be taken for granted (bona-fide assertions) whereas others as requiring explanations (we call them fiat assertions for brevity).

We impose restrictions on the choice of the explanations (Δs) computed during the abduction process. In particular, a scoring function S evaluates an explanation Δ according to the two criteria proposed by Thagard for selecting explanations [Thagard, 1978], namely simplicity and consilience. According to Thagard, the less hypothesized assertions an explanation contains (simplicity) and

[14] $\Leftarrow$ denotes the assignment operator

the more ground assertions (observations) an explanation involves (consilience), the higher its preference score. The following function can be used to compute the preference score for a given explanation[15]: $S(\Sigma, \mathcal{R}, \mathcal{A}, \Delta) := S_f(\Sigma, \mathcal{R}, \mathcal{A}, \Delta) - S_h(\Delta)$. The function S_f represents the number of assertions in the explanation (Δ) that follow from $\Sigma \cup \Delta$, and the function S_h represents the number of assertions in the explanation. Thus, S_f and S_h can be defined as follows:

$$S_f(\Sigma, \mathcal{R}, \mathcal{A}, \Delta) := \sharp\{\alpha \in \mathcal{A} \mid \Sigma \cup \Delta \models_{\mathcal{R}} \alpha\}$$
$$S_h(\Delta) := \sharp\Delta$$

The function $maximize(\Sigma, \mathcal{R}, \mathcal{A}, \Delta s, S)$ selects those explanations $\Delta \in \Delta$s for which the score $S(\Sigma, \mathcal{R}, \mathcal{A}, \Delta)$ is maximal, i.e., there exists no other $\Delta' \in \Delta s$ such that $S(\Sigma, \mathcal{R}, \mathcal{A}, \Delta') > S(\Sigma, \mathcal{R}, \mathcal{A}, \Delta)$.

At the end of the introduction to CAE we consider a small example. Consider the following ABox

$$\Gamma_0 = \{City(Hamburg), AirPollution(Hamburg)\}.$$

Furthermore, we consider the following set of abduction rules $\mathcal{R}_0$:

$$\{AirPollution(x) \leftarrow City(x), Near(x, y), Industry(y), \quad (15)$$
$$AirPollution(x) \leftarrow TrafficJam(x)\} \quad (16)$$

The abduction engine is started with $CAE(\Omega_0, \Xi_0, \emptyset, \mathcal{R}_0, S, \Gamma_0)$, where Ω_0 and Ξ_0 are default values for the selection strategy and the termination condition, respectively (not explained here). Given this call to CAE, the first explanation generated by *compute_explanation* is:

$$\Delta_1 = \{\textit{Near(Hamburg, Ind1), Industry(Ind1)}\} \quad (17)$$

where *Ind1* is a new constant generated by the interpretation engine. This explanation does not contain $City(Hamburg)$ since this is given in $\mathcal{A}$ (it need not be hypothesized). Similarly, the second explanation based on the second abduction rule is:

$$\Delta_2 = \{\textit{TrafficJam(Hamburg)}\} \quad (18)$$

In the first explanation Δ_1, there is one explicit assertion (*City(Hamburg)*) and two hypothesized assertions($Near(Hamburg, Ind1), Industry(Ind1)$). Consequently, the score of Δ_1 is calculated as follows:

$$S(\Sigma, \mathcal{R}, \Gamma_0, \Delta_1) = 0 - 2 = -2$$

Similarly, in Δ_2 there is no explicit assertion but one hypothesited assertion(*TrafficJam*(*Hamburg*)).

$$S(\Sigma, \mathcal{R}, \Gamma_0, \Delta_2) = 0 - 1 = -1$$

In this example, Δ_2 would be preferred. If there are two possible explanations with the same score one could, for instance, generate queries to KDMA or HCI to obtain more information for ranking the explanations. In this case, as a result the Abox
$\{AirPollution(Hamburg), City(Hamburg), TrafficJam(Hamburg)\}$ is returned.

[15]For the sake of brevity the parameters of S are not shown in the previous functions.

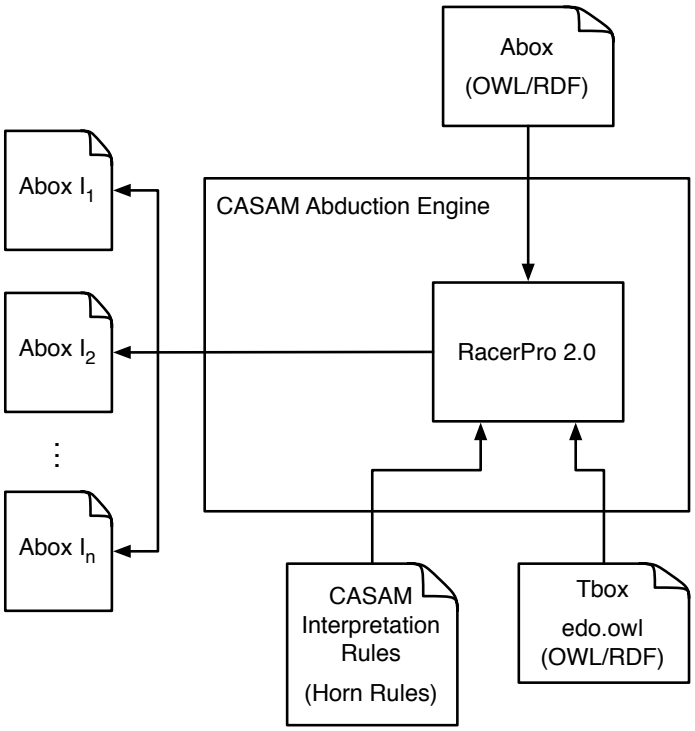

Figure 3: Interface of the CASAM abduction engine.

3.3.6 Interfaces to the CASAM Abduction Engine

The CASAM abduction engine (CAE) is developed on the basis of the abduction engine already used in the BOEMIE[16] project. The core component of the CAE is the RacerPro 2.0 reasoner and its abduction engine.

The CAE expects as input an Abox in the OWL/RDF format, which in our case is created from the output of the Alchemy Inference Module (see Figure 2). Also a Tbox and abduction rules are specified in order to adopt the engine to the domain of the project. Please note that both are stored in external files so that the domain independence of the engine is guaranteed.

As already mentioned, the abduction engine computes explanations for each assertion in the input Abox based on the possible explanations that are specified in the abduction rules. The explanation with the best score is added to the Abox. For all explanations with the same score there is a corresponding interpretation Abox and the computation continues with each interpretation Abox separately. Figure 3 shows the interface of the CASAM abduction engine with all its input and output files.

3.4 Complete Example

In this section, an example is given which goes step by step through the RMI engine. Assume RMI receives the next weighted Abox from KDMA component:

$$\Gamma_0 = \{0.2\ AirPollution(Hamburg), -0.1\ AirPollution(Berlin), 0.3\ TrafficJam(Berlin)\}$$

The following .mln file is used as input to the first Alchemy system which contains also the weighted assertions of the above Abox:

[16] www.boemie.org

$city = \{Hamburg, Berlin\}$

$TrafficJam\,(city)$
$AirPollution\,(city)$

$0.5\ TrafficJam(x) \Rightarrow AirPollution(x)$

$0.2\ AirPollution(Hamburg)$
$-0.1\ AirPollution(Berlin)$
$0.3\ TrafficJam(Berlin)$

Table 3: Input to the first Alchemy system

The first Alchemy system calculates the most probable world $\mathcal{W}$ based on the above .mln file. The output of this module is the following Abox Γ_0:

$AirPollution\,(Hamburg)$
$AirPollution\,(Berlin)$
$TrafficJam\,(Berlin)$
$\neg TrafficJam\,(Hamburg)$

Table 4: The most probable world $\mathcal{W}$ generated by the first Alchemy system

The next step computes additional support for these "percepts" by considering background knowledge. This process is performed in the next component of the RMI engine namely CAE. For the example we use the following abduction rules $\mathcal{R}$:

$$AirPollution(x) \leftarrow City(x), Near(x, y), Industry(y) \quad (19)$$
$$AirPollution(x) \leftarrow TrafficJam(x) \quad (20)$$

The rules are used in the non-deterministic CAE function *compute_explanation*.

We first assume that Ω picks $(\Gamma_0, AirPollution(Hamburg))$. Since there are two assertions for the *AirPollution* observation, there are two potential explanations (see the example above). As we have seen above, the explanation with $TrafficJam(Hamburg)$ would in principle be preferred, but it leads to an inconsistency. Thus, the other explanation is used and $\Gamma_1 := \Gamma_0 \cup \{Near(Hamburg, Ind1), Industry(Ind1)\}$. The function *explanation_step* returns $\{\Gamma_1\}$.

Then, let us assume that Ω selects $(\Gamma_1, AirPollution(Berlin))$. Given the examples above, it is not difficult to see that the second rule will lead to the preferred explanation. Actually, $TrafficJam(Berlin)$ is already true. Thus, finally CAE will return $\{\Gamma_1\}$ (we neglect the postprocessing step using Alchemy here).

3.5 Interfaces to RMI

In the CASAM project the communication architecture is based on inter-component webservice calls. Besides RMI there are three additional components, which together compose the CASAM system. These are the Human-Computer Interaction component (HCI), the Knowledge-Driven Multimedia Analysis component (KDMA), and the Integration Platform, abbreviated by IP. An overview of the provided and required interfaces, that are relevant for the first RMI prototype, is given in Figure 4. The fact that the coordination and distribution of messages (calls) is handled by a BPEL Engine, and therefore only indirect calls are possible, will be ignored in the further description.

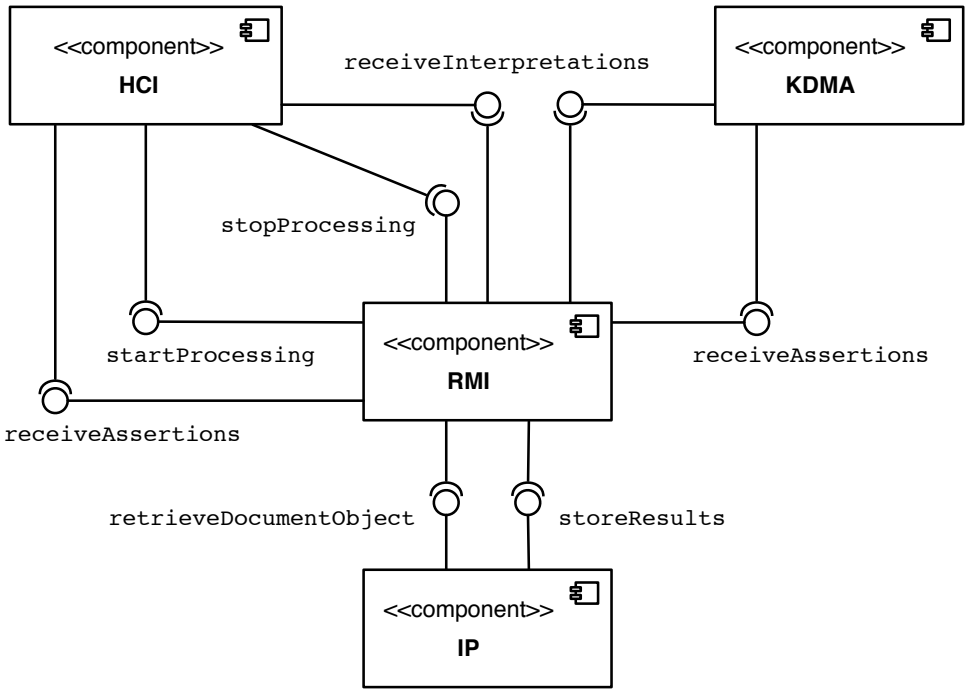

Figure 4: Provided and required interfaces of the RMI component

The interpretation process starts with a `startProcessing` call from HCI to RMI when a user selects a multimedia file for annotation. This call induces RMI to retrieve the required information about the multimedia file for the interpretation process from the Integration Platform by the use of the `retrieveDocumentObject` interface. These are, among other things, the Tbox and the Input Abox. In the meantime, KDMA starts the analysis which results in assertions that are generated and transmitted to RMI by the use of a `receiveAssertions` interface call (from either HCI or KDMA). Subsequently, RMI starts the interpretation pipeline. At the end of this process the created interpretations are made available to HCI and KDMA via the `receiveInterpretations` interface. When a user is satisfied with the interpretation results he can trigger a `stopProcessing` call from HCI using the graphical user interface. The last interface that RMI uses is responsible for the storage of the user-accepted interpretation results (Result Abox). This interface is provided by the Integration Platform and is addressed by `storeResults`.

4 Evaluation of the Basic RMI Engine

In order to get design decisions for the final reasoning engine for multimedia interpretation, the basic RMI engine is evaluated in the following. While Alchemy [Kok et al., 2005] has been chosen as the basic probabilistic reasoning engine due to its high optimizations in the Markov logic setting, CAE has been built up as a prototype based on the highly optimized RacerPro system. However, currently, the default setting for ranking interpretations is used (see above). In the future, scoring will be based on probabilistic reasoning. Since until now no role assertions have been provided by the multimedia analysis component KDMA, they have not been considered for the evaluation. In Section 4.1 issues regarding the efficiency of Alchemy and the CAE are discussed, and in Section 4.2 the quality of their results is discussed.

4.1 Efficiency

In a nutshell, the evaluation results show that the runtime performance of Alchemy concerning MAP basically satisfies the requirements of CASAM: Under realistic settings, results are computed in several seconds. This is, however, not yet the case for the CAE process.

Besides the MCMC algorithm of Alchemy, the performance of the computation of the most probable world depends on the expressivity of the Tbox $\mathcal{T}$, the number $|\mathcal{T}|$ of axioms to be considered, the number $|\mathcal{A}|$ of assertions and the number $|\mathcal{I}|$ of individuals. The expressivity of $\mathcal{T}$ is $\mathcal{ALCH}_f{}^-$, the language presented in Chapter 2.1. Remember that assertions and individuals representing the multimedia content have been filtered out for probabilistic reasoning. In accordance to the first analysis results of KDMA, only concept assertions are considered to belong to $|\mathcal{A}|$ in this setting. For the structure of $\mathcal{A}$, we consider the worst case in which every concept assertion is specified with a different individual such that $|\mathcal{A}| = |\mathcal{I}|$.

In order to further optimize the reasoning process, it is possible to consider only axioms of $\mathcal{T}$ being relevant for $\mathcal{A}$. Therefore we distinguish three cases: (i) axioms based on the signature of first analysis results of KDMA, (ii) axioms based on a possible average case signature of assertions[17] and (iii) the worst case in which the whole signature is required. Table 5 shows the development of the runtime performance of MAP with Alchemy for these cases by increasing the number of concept assertions with different individuals (but the signature of $\mathcal{A}$ remains the same). If no results were obtained after 300 seconds, this is indicated with $-$.

$\lvert\mathcal{A}\rvert = \lvert\mathcal{I}\rvert$	$\lvert\mathcal{T}\rvert$ w.r.t. first results	average $\lvert\mathcal{T}\rvert$	full $\lvert\mathcal{T}\rvert$
10	0.141	0.826	3.354
20	0.405	3.432	14.82
30	0.842	9.952	44.538
40	1.450	21.621	98.124
50	2.433	39.842	186.109
60	3.588	70.450	–
70	5.132	153.941	–

Table 5: MAP performance of Alchemy (measurements in seconds).

All tests were performed on a Pentium (R) dual-core 2.50 GHz CPU with 4.00 GB RAM on a 32 bit operating system. As can be seen from the results of the average case, the results could be improved for real-time scenarios, especially when lots of assertions resp. individuals are involved. After filtering out the MCO assertions, the initial analysis results (preprocessed Input Abox) consist of about 10 assertions with 10 individuals.

In Section 3.2 it was shown that in Alchemy it is possible to specify **typings** for predicates, i.e., the space of individuals variables of the corresponding predicates are allowed to be substituted with. Since this kind of restriction rules out lots of possible worlds, it is similar to the specification of strict axioms. In other words, typings provide an elegant means to achieve efficiency by exploiting strict domain knowledge (cf. Chapter 2.3). But the power of typing is restricted: While a variant

[17] A signature around four times smaller than the signature of the whole Tbox

of strict disjointness can be modeled with concept typings and because strict domain and range restrictions can be specified with role typings, there is no possibility to obtain the strictness of other ontology design patterns of $\mathcal{ALH}_f{}^-$. However, as can be seen from Table 6, concept and role typings improve the performance significantly:

$\lvert\mathcal{A}\rvert = \lvert\mathcal{I}\rvert$	$\lvert\mathcal{T}\rvert$ w.r.t. first results	average $\lvert\mathcal{T}\rvert$	full $\lvert\mathcal{T}\rvert$
10	0.093	0.327	1.232
20	0.249	0.936	3.010
30	0.468	2.340	5.803
40	0.780	4.555	18.252
50	1.263	8.221	32.448
60	1.762	13.228	52.463
70	2.636	20.155	77.828

Table 6: MAP performance of Alchemy with typings (measurements in seconds).

The results of the average case are quite satisfying, since MAP is computed in several seconds. But if there are lots of assertions, probabilistic reasoning still is not as fast as possibly expected by the users of the system.

The performance of CAE basically depends on the expressivity of $\mathcal{T}$, the number $|\mathcal{T}|$ of axioms, the number $|\mathcal{R}|$ and complexity of abduction rules and the number $|\mathcal{A}|$ of assertions. While $|\mathcal{A}|$ and $|\mathcal{R}|$ determine the number of abduction rules applicable (the number of assertions explainable with abduction rules), the expressivity and size of $\mathcal{T}$ as well as the complexity of the rules determine the runtime performance for each explanation. The performance of the current abduction algorithm also depends on the order and size of the rules.

The input to CAE is the most probable world derived from Alchemy under consideration of typings. We distinguish two cases: (i) three rather simple abduction rules and (ii) eight more complex abduction rules. In Table 7 the runtime performance of the interpretation engine is shown in seconds.

$\lvert\mathcal{A}\rvert = \lvert\mathcal{I}\rvert$	$\lvert\mathcal{R}\rvert = 3$	$\lvert\mathcal{R}\rvert = 8$
10	1.090	4.897
20	1.609	89.478
30	2.180	-

Table 7: Performance of the interpretation engine (measurements in seconds).

Results were presented in order to show the high complexity of multimedia interpretation algorithms: Remember that the interpretation engine has been adapted from the BOEMIE project. Interpretations are chosen according to the preference score presented in Section 3.3.5. After applying the *cutoff-criterion*, a dynamic optimization ruling out all intermediate interpretations known to provide no better score by applying further explanations, the runtime performance has been reduced from hours to minutes. However, as can be seen from Table 7, these optimizations do not satisfy the runtime requirements of CASAM, since the method does not scale up if there are lots of rules. But as mentioned in the beginning of the chapter, the objective of this version of the interpretation engine was to simply provide first results rather than to do this in a satisfying amount of time.

4.2 Quality

The most probable explanation in all test cases has been computed correctly. Under slight changes of the MLN file, the most probable world is also as expected. Based on the most-probable explanation and the current preference score, the basic interpretation engine of CASAM provides correct results. However, the results are incomplete with respect to several aspects.

Though it is possible to specify role typings (i.e., domain and range restrictions for roles) in the MLN files of Alchemy, there is no possibility to specify typing constraints for specific tuples. As a consequence, it is possible that there are role assertions in the most probable world for which there is no information at all.

Consider the role *interviews* which is domain restricted with *Interviewer* and range restricted with *Interviewee*. Let $interviewer = \{ind_1, ind_2\}$ and $interviewee = \{ind_3\}$ be the corresponding types. Possible ground atoms for *interviews* then are restricted to $interviews(ind_1, ind_3)$ and $interviews(ind_2, ind_3)$. If there is no information about these atoms, they are true and false both with probability 0.5. Since there is no preference, MAP has to choose at random such that the most-probable world probably contains one or both of the atoms.

To overcome this problem, all roles are assigned with a marginally negative weight in the MLN file. Then, the quality of the MAP algorithm of Alchemy is satisfying: In all test cases applied, the most-probable world is computed correctly and as expected. Further it is materialized, i.e., it does contain all implicit assertions.

However, there are some drawbacks with MAP in general: Assume KDMA provides the assertion $0.51\, Car(ind_1)$ and there is no other information regarding ind_1. Then $Car(ind_1)$ is in the most probable world, but KDMA nearly is guessing this assertion (due to the very low corresponding weight of approximately 0.04). Further, there is no difference if KDMA instead provides the assertion $0.91\, Car(ind_1)$, i.e., the highest probability is relevant, but not the probability itself. Additionally, in the case of conflicts such as $\{0.91\, Car(ind_1), 0.89\, Forest(ind_1), Car \sqsubseteq \neg Forest\}$, the less probable alternative is not considered at all.

The basic RMI engine can be used to build an initial prototype of the CASAM system, but more research is required to satisfy the real-time requirements coming from the human-in-the-loop scenario that is foreseen in CASAM.

5 Conclusion and Outlook

We have described the architecture and modules of the basic RMI engine. An evaluation has been carried out with artificial datasets. The evaluation shows that, despite that also probabilistic knowledge is considered in CASAM, the performance characteristics are promising. Optimizations will be required for the CAE module. In particular, we would like to avoid the generation of interpretation Aboxes that will be eliminated later on. This will be achieved by a tighter integration of the second probabilistic module as a preference score into the CAE module. Furthermore, it might happen that the best "world" computed by the first probabilistic module turns out not to lead to satisfactory interpretations. If this turns out to be a problem, the second-best world will also have to be tried et cetera. These issues will be investigated in the future.

References

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F. (January 2003). *The Description Logic Handbook: Theory, Implementation and Application*. Cambridge UP: Cambridge, NY.

[Baader and Hanschke, 1991] Baader, F. and Hanschke, P. (1991). A scheme for integrating concrete domains into concept languages. *International Conference on Artificial Intelligence*, page 452457.

[Castano et al., 2008] Castano, S., Espinosa, S., Ferrara, A., Karkaletsis, V., Kaya, A., Möller, R., Montanelli, S., Petasis, G., and Wessel, M. (2008). Multimedia interpretation for dynamic ontology evolution. In *Journal of Logic and Computation*. Oxford University Press.

[Domingos and Richardson, 2007] Domingos, P. and Richardson, M. (2007). Markov logic: A unifying framework for statistical relational learning. In Getoor, L. and Taskar, B., editors, *Introduction to Statistical Relational Learning*, pages 339–371. Cambridge, MA: MIT Press.

[Gilks et al., 1996] Gilks, W. R., Richardson, S., and Spiegelhalter, D. J. (1996). *Markov Chain Monte Carlo in Practice*. Chapman and Hall, London, UK.

[Kok et al., 2005] Kok, S., Singla, P., Richardson, M., and Domingos, P. (2005). The alchemy system for statistical relational ai. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA.

[Neal, 1993] Neal, R. M. (1993). Probabilistic inference using markov chain monte carlo methods. Technical report, University of Toronto, Dept. of Computer Science.

[Pearl, 1988] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA.

[Russell and Norvig, 2003] Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall.

[Thagard, 1978] Thagard, R. P. (1978). The best explanation: Criteria for theory choice. *The Journal of Philosophy*.

A Installation and configuration instructions for RMI

- Content of the archive file RMI.zip:
  - RMI-code.zip (RMI Java code)
  - Rules.zip (demo interpretation rules)
  - Ontologies.zip (current EDO and MCO ontologies)
  - CASAM-RacerPro-2-0-Preview-Package.zip
    (RacerPro 2.0 preview version - licensed until the end of 2011)

  All of the files are provided within the accompanying archive file and are required to run RMI. Additionally, the following software should be installed on the system, RMI will be executed on:
  - Eclipse IDE for Java EE developers (http://www.eclipse.org)
  - Sun Java 6 (http://www.java.com)
  - Alchemy (http://alchemy.cs.washington.edu)
  - optional: Tomcat 6.0.20 (http://tomcat.apache.org)

- Installation:

  Extract the RMI.zip file. The file was generated directly out of Eclipse. To install the software, create a new "Dynamic Web Project" named "RMI". While the newly created Project is still highlighted, go to "File → Import" and then choose "Archive File". Follow the wizard to import the code into the RMI project.

- Configuration:
  1. Once the project was successfully imported, it should be checked if there are any errors that occur (white crosses on red ground). Because all the configuration settings are included in the archive file, Eclipse maybe asks for a local Tomcat installation. If this is the case, a Tomcat (6.0.20) must be downloaded and installed in the Eclipse environment. Otherwise, the configuration can continue with the next step.
  2. There are some files, that have to be adapted to the local machine:
     (a) `de.tuhh.sts.biws.ConfigurationFinder.java`
     Replace

     config = new XMLConfiguration
     ("/Applications/eclipse/workspace/RMI/WebContent/WEB-INF/conf/biws_local.xml");

     in line 26 with the path to where the configuration file is located on the local machine.
     (b) `RMI/WebContent/WEB-INF/conf/biws_local.xml`
     Go through the configuration xml file and change the paths accordingly (e.g. the path to where RacerPro was installed to, etc.). Note that an "input" and an "output" folder should be created somewhere on the system before running RMI for the first time. The paths to those folders should be given in line 16 and 17. Under <racerservers> you can configure two instances of RacerPro that will be required for processing. Ensure that the ports are not conflicting with any other services on the machine. Unpack Rules.zip to a folder of your choise and change the path under <ruleSet>. The next step would be to unpack the Ontologies.zip file to a folder of your choise. It contains ontologies in different formats that are required for the first prototype and will be negligible in the near future. The last modification that has to be made is the path of the racer mirror file. Therefore a file called init.racer has to be created and it has to be filled with the following content,

(mirror "http://www.casam-project.eu/edo.owl"
"file:///Users/Nummer5/Desktop/ontologies/edo/edo_owlrdf.owl")
(mirror "http://www.casam-project.eu/mco.owl"
"file:///Users/Nummer5/Desktop/ontologies/mco/mco.owl")

where the paths point to the folder you have chosen for the ontologies.

(c) `de.tuhh.sts.rmi.alchemy.AlchemyFactory.java`
The last changes have to be made in the `AlchemyFactory` class. All the paths, defined at the beginning of the class, have to be set correctly. In line 93, a file called `empty.db` for Alchemy is referenced; it has to be created before running RMI for the first time.

- Running RMI:

1. The class that should be used to run RMI is also the `AlchemyFactory.java` which was modified in the previous step. It contains a main class which starts the processing taking some sample data as input . If all the installation and configuration work was successful, RMI should start the processing now.