Integrating Concept-Based Knowledge Bases

Aida Vitória and **Margarida Mamede**
Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
2825–114 Caparica, Portugal
{av,mm}@di.fct.unl.pt

1 Introduction

The main goal of this paper is to propose a basic set of compositional operators that can be used to combine separate concept-based knowledge bases. In particular, we focus our attention on the concept language $\mathcal{AL}$ extended with a restricted form of terminological axioms. We have chosen $\mathcal{AL}$ because, to the extent of our knowledge, the problem of integrating concept-based knowledge bases has not been addressed yet, and we wanted to start studying it with a rather simple and well-understood language. In this context, $\mathcal{AL}$ seems to be a natural choice, since it can be considered as a core language of a family of well-studied concept languages [10]. We plan to extend our work to more expressive concept languages, which are also more suitable for practical applications, but this is left for future work.

In [8, 9], concept languages are used to integrate information in the context of databases, where both schema and data integration are important issues. In this paper, we are concerned with information integration in the area of knowledge representation. Concept-based knowledge bases are seen as logical theories that can be merged (or composed). To this end, we propose three basic operations for composing knowledge bases: *union* ($\vee$), *intersection* ($\wedge$) and *renaming* (RN).

We characterize the above operators in two distinct ways. Firstly, we define a semantics that abstracts away from implementation details. Secondly, we present an implementation for $\mathcal{AL}$-knowledge bases, which transforms compositions of knowledge bases into knowledge bases. This transformation is sound and complete with respect to the semantics referred above.

It is worth referring that compositional operators have also been investigated in the context of logic programming [5, 6, 7, 13, 14], for combining not only definite programs [5], but also general logic programs [3, 4].

2 The language $\mathcal{AL}$

The language used in this work is $\mathcal{AL}$ [10] extended with terminological axioms of the form $R_1 \sqsubseteq R_2$, where R_1 and R_2 are primitive relations. Even though it is an extension of $\mathcal{AL}$, for the sake of simplicity, we still call it $\mathcal{AL}$.

Let (Conc, Rel, Ind) be a tuple of pairwise disjoint sets of *primitive concepts*, Conc, *primitive binary relations*, Rel, and *individuals*, Ind. In the sequel, we assume that $A \in \text{Conc}$, $R \in \text{Rel}$, and $i \in \text{Ind}$. A *concept* C and an *assertion* β are defined by the following grammar:

$$\begin{array}{lcl} C & \longrightarrow & \top \mid \bot \mid A \mid \neg A \mid C_1 \sqcap C_2 \mid \forall R.C \mid \exists R \\ \beta & \longrightarrow & i : C \mid R(i_1, i_2) \mid R_1 \sqsubseteq R_2 \ . \end{array}$$

A *knowledge base* Σ is a finite set of assertions.

Given an alphabet (Conc, Rel, Ind), an *interpretation* of $\mathcal{AL}$ is a pair $I = (\Delta^I, \cdot^I)$, where Δ^I is a nonempty set, called the *interpretation domain*, and $\cdot^I$ is the *interpretation function* which satisfies the following conditions: $i^I \in \Delta^I$, for every $i \in \text{Ind}$; $A^I \subseteq \Delta^I$, for every $A \in \text{Conc}$; and $R^I \subseteq \Delta^I \times \Delta^I$, for every $R \in \text{Rel}$. The interpretation function is extended to arbitrary concepts in the usual way. An interpretation I *satisfies* an assertion $i : C$ iff $i^I \in C^I$, $R(i_1, i_2)$ iff $(i_1^I, i_2^I) \in R^I$, and $R_1 \sqsubseteq R_2$ iff $R_1^I \subseteq R_2^I$.

A knowledge base Σ is *satisfiable* if there is an interpretation I that satisfies all of its assertions. In this case, I is said to be a *model* of Σ. When Σ is not satisfiable, we say that Σ is *unsatisfiable.* Two knowledge bases are *equivalent* iff they have the same models. Furthermore, an assertion β is a *logical consequence* of a knowledge base Σ, written $\Sigma \models \beta$, if every model of Σ satisfies β.

It is easy to verify that every $\mathcal{AL}$-knowledge base can be rewritten into an equivalent $\mathcal{AL}$-knowledge base in which there are no concept conjunctions. This follows from the equality

$$(\forall R.(C_1 \sqcap C_2))^I = ((\forall R.C_1) \sqcap (\forall R.C_2))^I,$$

for every interpretation I, and from the fact that knowledge bases $\{i : C_1 \sqcap C_2\}$ and $\{i : C_1, i : C_2\}$ are equivalent. Thus, to simplify the rest of the work, we will assume that assertions and knowledge bases do not comprise any sub-concept of the form $C_1 \sqcap C_2$.

The algorithm for checking whether a knowledge base is satisfiable and our implementation of the compositional operators rely on the notion of completion of a knowledge base. Thus, let us consider another language of assertions, $\mathcal{AL}^+$, over the alphabet (Conc, Rel, Ind, Var), where Var is a set of *variables* ($v \in \mathrm{Var}$), disjoint from Conc, Rel, and Ind. The definition of $\mathcal{AL}^+$-assertion is similar to the definition of $\mathcal{AL}$-assertion, except that variables can occur in any place individuals may appear. Because of this, a *term* t designates either an individual or a variable, and let $\mathrm{Term} = \mathrm{Ind} \cup \mathrm{Var}$. For the sake of conciseness, $\mathcal{AL}$-assertions and $\mathcal{AL}^+$-assertions will sometimes be called assertions. The definition of interpretation $I = (\Delta^I, \cdot^I)$ is extended in order to associate an element $v^I \in \Delta^I$ with every variable v.

The procedure that builds a completion of an $\mathcal{AL}$-knowledge base Σ works by adding new $\mathcal{AL}^+$-assertions to Σ, according to the following five rules, which may be applied in any order, until no rule is applicable. The final set of assertions is called a *completion*[1] of Σ and is denoted by $\overline{\Sigma}$.

($\exists$) **If** $t_1 : \exists R \in \overline{\Sigma}$
and there is no term t_2 s.t. $R(t_1, t_2) \in \overline{\Sigma}$,
then $\overline{\Sigma} \leftarrow \overline{\Sigma} \cup \{R(t_1, v)\}$,
where v is a new variable.

(R) **If** $R(t_1, t_2) \in \overline{\Sigma}$
and $t_1 : \exists R \notin \overline{\Sigma}$,
then $\overline{\Sigma} \leftarrow \overline{\Sigma} \cup \{t_1 : \exists R\}$.

($\sqsubseteq_+$) **If** $R_1 \sqsubseteq R_2 \in \overline{\Sigma}$, $R_2 \sqsubseteq R_3 \in \overline{\Sigma}$
and $R_1 \sqsubseteq R_3 \notin \overline{\Sigma}$,
then $\overline{\Sigma} \leftarrow \overline{\Sigma} \cup \{R_1 \sqsubseteq R_3\}$.

($\sqsubseteq_R$) **If** $R_1 \sqsubseteq R_2 \in \overline{\Sigma}$, $R_1(t_1, t_2) \in \overline{\Sigma}$
and $R_2(t_1, t_2) \notin \overline{\Sigma}$,
then $\overline{\Sigma} \leftarrow \overline{\Sigma} \cup \{R_2(t_1, t_2)\}$.

($\forall_R$) **If** $t_1 : \forall R.C \in \overline{\Sigma}$, $R(t_1, t_2) \in \overline{\Sigma}$
and $t_2 : C \notin \overline{\Sigma}$,
then $\overline{\Sigma} \leftarrow \overline{\Sigma} \cup \{t_2 : C\}$.

A similar completion algorithm was introduced in [11] for the language $\mathcal{ALC}$.

Proposition 1 *Let Σ be an $\mathcal{AL}$-knowledge base. The procedure that computes a completion of Σ always terminates.*

A knowledge base Σ has a *clash* if it contains either an assertion of the form $t : \bot$, or two assertions of the form $t : A$ and $t : \neg A$. Otherwise, Σ is *clash-free*.

[1] Completions are not unique (even if we consider the equivalence generated by variable renaming). For instance, two possible completions of $\Sigma = \{i : \exists R_1, i : \exists R_2, R_1 \sqsubseteq R_2\}$ are $\{R_1(i, v_1), R_2(i, v_2), R_2(i, v_1)\}$ and $\{R_1(i, v), R_2(i, v)\}$.

Proposition 2 *Let Σ be an $\mathcal{AL}$-knowledge base and $\overline{\Sigma}$ be a completion of Σ. Then, Σ is satisfiable if and only if $\overline{\Sigma}$ is clash-free.*

3 Combining Knowledge Bases

To begin with, let us present the declarative semantics of the compositional operators: *union* ($\vee$), *intersection* ($\wedge$), and *renaming* (RN).

In the sequel, we use Σ to designate both an $\mathcal{AL}$-knowledge base and its name. Besides, a *renaming function* ρ is a total function from $\mathrm{Conc} \cup \mathrm{Rel}$ to a set of primitive concepts and primitive relations, $\mathrm{Conc}' \cup \mathrm{Rel}'$, such that: $\mathrm{Conc}' \cap \mathrm{Rel}' = \mathrm{Conc} \cap \mathrm{Rel}' = \mathrm{Conc}' \cap \mathrm{Rel} = \emptyset$.

Starting from a collection of $\mathcal{AL}$-knowledge bases Σ and a collection of renamings ρ, we define the language of *$\mathcal{AL}$-knowledge base expressions E* as follows:

$$E \longrightarrow \Sigma \mid RN(E, \rho) \mid E_1 \vee E_2 \mid E_1 \wedge E_2 .$$

The effect of renaming a knowledge base Σ via ρ ($RN(\Sigma, \rho)$) is that every occurrence in Σ of a primitive concept $A \in \mathrm{Conc}$ (resp., primitive relation $R \in \mathrm{Rel}$) is replaced by $\rho(A)$ (resp., $\rho(R)$). Obviously, the renaming has no effect if ρ is the identity function. For this reason, we write $\rho = [d_1/d'_1, \ldots, d_n/d'_n]$ (where $n \geq 1$) to make explicit that:

- $\rho(d_i) = d'_i$, for all $i = 1, \ldots, n$; and
- $\rho(d) = d$, for every $d \notin \{d_1, \ldots, d_n\}$.

An $\mathcal{AL}$-knowledge base expression E is characterized in terms of a set of $\mathcal{AL}$-assertions, by induction on the structure of E, in the following way:

$$\begin{aligned} RN(E, \rho) &= \rho(\{\beta \mid E \models \beta\}) \\ E_1 \vee E_2 &= \{\beta \mid E_1 \models \beta \text{ or } E_2 \models \beta\} \\ E_1 \wedge E_2 &= \{\beta \mid E_1 \models \beta \text{ and } E_2 \models \beta\} . \end{aligned}$$

Remark that, according to the semantics given before, an $\mathcal{AL}$-knowledge base Σ is equivalent to the set

$$\{\beta \mid \Sigma \models \beta\}.$$

If $\mathcal{M}(\Sigma)$ denotes the set of models of an $\mathcal{AL}$-knowledge base Σ, one could be lead to believe that

$$\mathcal{M}(\Sigma_1 \vee \Sigma_2) = \mathcal{M}(\Sigma_1) \cap \mathcal{M}(\Sigma_2)$$

and

$$\mathcal{M}(\Sigma_1 \wedge \Sigma_2) = \mathcal{M}(\Sigma_1) \cup \mathcal{M}(\Sigma_2).$$

To understand why this is not the case, let

$$\Sigma_1 = \{R(i_1, i_2)\} \text{ and } \Sigma_2 = \{R(i_1, i_3)\}.$$

Then, $\Sigma_1 \wedge \Sigma_2 = \{i_1 : \exists R\}$, and the interpretation M such that

$$\Delta^M = \{i_1^M, i_2^M, i_3^M, a\} \quad \text{and} \quad R^M = \{(i_1^M, a)\}$$

is a model of $\Sigma_1 \wedge \Sigma_2$. Nevertheless, $M \notin \mathcal{M}(\Sigma_1) \cup \mathcal{M}(\Sigma_2)$.

Example 1 Let Σ_{fam} and $\Sigma_{disease}$ be the two following knowledge bases.

Σ_{fam}	$\Sigma_{disease}$
$\texttt{father}(\texttt{Joe},\texttt{Fred})$	$\texttt{Joe} : \forall\texttt{parent.diseaseA}$
$\texttt{mother}(\texttt{Mary},\texttt{Fred})$	$\texttt{Fred} : \forall\texttt{parent.diseaseA}$
$\texttt{father}(\texttt{Fred},\texttt{Rose})$	$\texttt{Mary} : \neg\texttt{diseaseA}$
$\texttt{Fred} : \forall\texttt{father.female}$	

In order to obtain information about other people infected with diseases, we could combine Σ_{fam} and $\Sigma_{disease}$ in any of the two following ways.

1. $RN(\Sigma_{fam}, \rho) \ \vee \ \Sigma_{disease}$, where $\rho = [\texttt{mother}/\texttt{parent}, \texttt{father}/\texttt{parent}]$.

2. $\Sigma_{fam} \vee \Sigma_{disease} \vee \Sigma_{par}$, where Σ_{par} relates `mother` and `father` to `parent`:

$$\texttt{mother} \sqsubseteq \texttt{parent} \qquad \texttt{father} \sqsubseteq \texttt{parent}.$$

In both cases, it is possible to conclude that `Fred` and `Rose` are infected with `diseaseA`. However, in the first case, family relations, like `mother` and `father`, are destroyed (e.g., $RN(\Sigma_{fam}, \rho) \ \vee \ \Sigma_{disease} \not\models \texttt{father}(\texttt{Joe},\texttt{Fred})$), whereas this kind of information is preserved in $\Sigma_{fam} \vee \Sigma_{disease} \vee \Sigma_{par}$. It is worth observing that renamings change the set of logical consequences of the knowledge bases, while unions augment the set of logical consequences of the knowledge bases involved. $\Diamond$

The previous example shows that the renaming operator can be used for two distinct purposes: to eliminate vocabulary differences from independently developed knowledge bases, and to implement some form of information protection policies.

The next step is to discuss a possible implementation of the proposed operations. We define a syntactic transformation τ that, given an $\mathcal{AL}$-knowledge base expression, produces an $\mathcal{AL}$-knowledge base. This technique is comparable with the compilation-oriented implementation of the language of definite logic programs expressions presented in [5]. Another alternative (not explored in this paper) is to implement the language of $\mathcal{AL}$-knowledge base expressions without building any concrete knowledge base. The latter approach would correspond to an interpretation-oriented implementation.

The definition of $\tau(E_1 \wedge E_2)$ relies on several notions. First of all, we need to compute completions of $\tau(E_1)$ and $\tau(E_2)$.

We also need the set $\text{Univ}(\Sigma)$ of universally quantified $\mathcal{AL}$-assertions derived from Σ. Precisely, $\text{Univ}(\Sigma)$ is the least set such that (where $n \geq 1$):

(Σ) **If** $i : \forall R.C \in \Sigma$, **then** $i : \forall R.C \in \text{Univ}(\Sigma)$;

($\sqsubseteq_\forall$) **If** $i : \forall R_1.\cdots.\forall R_n.C \in \text{Univ}(\Sigma)$ and $R \sqsubseteq R_n \in \Sigma$, **then** $i : \forall R_1.\cdots.\forall R.C \in \text{Univ}(\Sigma)$;

($\sqsubseteq_\exists$) **If** $i : \forall R_1.\cdots.\forall R_n.\exists R \in \text{Univ}(\Sigma)$ and $R \sqsubseteq R' \in \Sigma$, **then** $i : \forall R_1.\cdots.\forall R_n.\exists R' \in \text{Univ}(\Sigma)$;

($\forall_\neg$) **If** $i : \forall R_1.\cdots.\forall R_n.A \in \text{Univ}(\Sigma)$ and $i : \forall R_1.\cdots.\forall R_n.\neg A \in \text{Univ}(\Sigma)$, **then** $i : \forall R_1.\cdots.\forall R_n.\bot \in \text{Univ}(\Sigma)$; and

($\forall_\bot$) **If** $i : \forall R_1.\cdots.\forall R_n.\forall R.\bot \in \text{Univ}(\Sigma)$ and $i : \forall R_1.\cdots.\forall R_n.\exists R \in \text{Univ}(\Sigma)$, **then** $i : \forall R_1.\cdots.\forall R_n.\bot \in \text{Univ}(\Sigma)$.

Notice that, although rules ($\forall_\neg$) and ($\forall_\bot$) are not usually found in most of the concept-based knowledge base systems presented in the literature, very similar rules have already been used to solve completely different problems (c.f. [1, 2, 12, 16]).

The idea is to consider the $\mathcal{AL}$-assertions that belong to the completions or to the sets of universally quantified $\mathcal{AL}$-assertions derived from the completions. Thus, if $\Sigma_{\mathcal{AL}}$ stands for the set of $\mathcal{AL}$-assertions that belong to an $\mathcal{AL}^+$-knowledge base Σ, let

$$[\![\Sigma]\!] = \Sigma_{\mathcal{AL}} \cup \text{Univ}(\Sigma).$$

Finally, let Σ_1 and Σ_2 be two $\mathcal{AL}$-knowledge bases. We define:

$$\begin{aligned} \Sigma_1 \ll \Sigma_2 &= \{i : \forall R_1.\cdots.\forall R_n.C \in \Sigma_1 \mid \\ &\qquad\qquad i : \forall R_1.\cdots.\forall R_n.\bot \in \Sigma_2\}; \\ \Sigma_1 \nabla \Sigma_2 &= (\Sigma_1 \cap \Sigma_2) \cup (\Sigma_1 \ll \Sigma_2) \cup (\Sigma_2 \ll \Sigma_1). \end{aligned}$$

It is easy to verify that $\Sigma_1 \models \beta$ and $\Sigma_2 \models \beta$, for every $\mathcal{AL}$-assertion $\beta \in \Sigma_1 \nabla \Sigma_2$.

The transformation τ can now be defined, by induction on the structure of the knowledge base expressions:

$$\begin{aligned} \tau(\Sigma) &= \Sigma \\ \tau(RN(E, \rho)) &= \rho(\tau(E)) \\ \tau(E_1 \vee E_2) &= \tau(E_1) \cup \tau(E_2) \\ \tau(E_1 \wedge E_2) &= \end{aligned}$$

$$= \begin{cases} [\![\overline{\tau(E_1)}]\!] \nabla [\![\overline{\tau(E_2)}]\!] & \text{if } \overline{\tau(E_1)} \text{ and } \overline{\tau(E_2)} \\ & \text{are both clash-free} \\ \tau(E_1) & \text{if } \overline{\tau(E_1)} \text{ is clash-free} \\ & \text{and } \overline{\tau(E_2)} \text{ is not} \\ \tau(E_2) & \text{if } \overline{\tau(E_2)} \text{ is clash-free} \\ & \text{and } \overline{\tau(E_1)} \text{ is not} \\ \{i : \bot\} & \text{otherwise.} \end{cases}$$

Proposition 3 *An $\mathcal{AL}$-assertion β is a logical consequence of a knowledge base expression E if and only if β is a logical consequence of the knowledge base $\tau(E)$.*

The proofs of the propositions stated in this paper are presented in [15].

4 Conclusions and Future Work

The main goal of this paper was to introduce three operators (union, intersection and renaming) for composing knowledge bases expressed in a concept language. To this end, knowledge base expressions have been defined with a compositional semantics. A computational interpretation of knowledge base expressions has also been formalized through a transformation that maps knowledge base expressions into knowledge bases.

Concerning future work, we aim at investigating some of the following open problems.

On the one hand, it is possible to extend both the underlying description logic and the set of compositional operators. For instance, we could allow numeric restrictions or transitive roles and consider unions with overriding.

On the other hand, it is interesting to study alternative implementations. In this work, we have proposed a compilation oriented approach. Another possible implementation could be interpretation oriented so as not to need a new knowledge base to be built. The main advantage of this setting is that we could cater for distributed concept-based knowledge bases. In addition, it would be desirable to devise an algebra of knowledge base expressions. For example, having detected that $\Sigma_1 \wedge \Sigma_1$ and Σ_1 are equivalent knowledge base expressions, we could replace the latter by the former in order to gain efficiency in the computations.

Another open problem is how to obtain useful information from inconsistent theories. In the framework of combining separate knowledge bases, we should try to reconcile contradictions so that new reasonable conclusions may be derived. For instance, it seems natural to put together the knowledge of different experts in order to derive new knowledge that none of them had before. However, the experts might hold contradictory views about the problem.

Finally, another interesting issue is to find a declarative semantics of the compositional operators based on sets of models. We conjecture that, to achieve that goal, the concept language has to be more expressive.

References

[1] A. Borgida. From type systems to knowledge representation: Natural semantics specifications for description logics. *International Journal of Intelligent and Cooperative Information Systems*, 1(1):93–126, 1992.

[2] A. Borgida and P. F. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. *Journal of Artificial Intelligence Research*, 1:277–308, 1994.

[3] A. Brogi, S. Contiero, and F. Turini. Composing general logic programs. In J. Dix, U. Furbach, and A. Nerode, editors, *Proc. of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97)*, volume 1265 of *Lecture Notes in Artificial Intelligence*, pages 273–288. Springer-Verlag, 1997.

[4] A. Brogi, S. Contiero, and F. Turini. The use of renaming in composing general programs. In P. Flener, editor, *Proc. of the 8th International Workshop on Logic Programming Synthesis and Transformation (LOPSTR'98)*, volume 1559 of *Lecture Notes in Computer Science*, pages 124–142. Springer-Verlag, 1999.

[5] A. Brogi, P. Mancarella, D. Pedreschi, and F. Turini. Modular logic programming. *ACM Transactions on Programming Languages and Systems*, 16(4):1361–1398, 1994.

[6] A. Brogi and F. Turini. Fully abstract compositional semantics for an algebra of logic programs. *Theoretical Computer Science*, 149(2):201–229, 1995.

[7] M. Bugliesi, E. Lamma, and P. Mello. Modularity in logic programming. *Journal of Logic Programming*, 19&20:443–502, 1994.

[8] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Source integration in data warehousing. Technical Report DWQ-UNIROMA-002, DWQ Consortium, 1997.

[9] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Description logic framework for information integration. In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *Proc. of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 2–13. Morgan Kaufmann, 1998.

[10] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. *Information and Computation*, 134(1):1–58, 1997.

[11] B. Hollunder, W. Nutt, and M. Schmidt-Schauß. Subsumption algorithms for concept description languages. In L. C. Aiello, editor, *Proc. of the 9th European Conference on Artificial Intelligence (ECAI-90)*, pages 348–353. Pitman, 1990.

[12] M. Mamede and L. Monteiro. Towards complete answers in concept languages. In M. Filgueiras and L. Damas, editors, *Progress in Artificial Intelligence: Proc. of the 6th Portuguese Conference on Artificial Intelligence (EPIA'93)*, volume 727 of

Lecture Notes in Artificial Intelligence, pages 121–135. Springer-Verlag, 1993.

[13] D. Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, 6(1–2):79–108, 1989.

[14] L. Monteiro and A. Porto. Contextual logic programming. In G. Levi and M. Martelli, editors, *Proc. of the 6th International Conference on Logic Programming (ICLP'89)*, Series in Logic Programming, pages 284–302. MIT Press, 1989.

[15] A. Vitória and M. Mamede. Integrating concept–based knowledge bases. Technical Report TR UNL–DI–02–99, Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia, 1999.

[16] A. Vitória, M. Mamede, and L. Monteiro. The retrieval problem in a concept language with number restrictions. In C. Pinto-Ferreira and N. J. Mamede, editors, *Progress in Artificial Intelligence: Proc. of the 7th Portuguese Conference on Artificial Intelligence (EPIA'95)*, volume 990 of *Lecture Notes in Artificial Intelligence*, pages 69–82. Springer-Verlag, 1995.