A Unified Architecture for Knowledge Representation based on Description Logics [1]

Liviu Badea
AI Research Department
Research Institute for Informatics
8-10 Averescu Blvd., Bucharest, Romania
e-mail: badea@roearn.ici.ro

Abstract. This paper presents a *unified* architecture for knowledge representation based on description (terminological) logics. The novelty of our approach consists in trying to use description logics not only for representing the domain knowledge, but also for describing beliefs, epistemic operators and actions of intelligent agents in a unitary framework. For this purpose, we have chosen a decidable terminological language, called $\mathcal{ALC}^\star$, whose expressivity is high enough to be able to represent actions and epistemic operators corresponding to the majority of modal logics of knowledge and belief.

Additionally, we describe practical inference algorithms for the language $\mathcal{ALC}^\star$ which lies at the heart of our $\mathcal{RegAL}$[2] knowledge representation system. The algorithms are sound and *complete* and can be used directly for deciding the validity and satisfiability of formulas in the propositional dynamic logic (PDL) by taking advantage of the correspondence between PDL and certain terminological logics.

1 Description logics

Description logics[3] (DLs) are descendants of the famous KL-ONE language [3] and can be viewed as formalizations of the frame-based knowledge representation systems.

The relationship between DLs and logic is analogous to the relationship between structured and unstructured programming languages. Indeed, DLs impose a certain discipline in the logical structure of a formula (concept) in the very same way in which the structured programming paradigm imposes a discipline in the control structure of a program. Although they somehow restrict the expressivity of the description language, DLs are most of the time preferable to general logic because of their increased understandability and usability in building practical knowledge bases. Also, as opposed to general logic, certain DLs may possess *decidable* inference problems while retaining a fairly high expressivity which enables them to represent complex ontologies.

Practically all the terminological knowledge representation systems built before 1990 (such as KL-ONE, KRYPTON, LOOM, BACK) used *incomplete* inference algorithms, the problem of finding complete algorithms being non-trivial for most of the interesting languages. This pragmatic approach of using incomplete inference algorithms has certain important drawbacks, the main one being the contradiction between the theoretically formalized semantics of the language (which is clear and easily communicable to a user) and the procedural semantics of the incomplete inference engine, the multiple cases of incompleteness being usually very difficult to explain to a user who is not supposed to know all the details of the implementation. This is why efforts have been invested in developing $\mathcal{RegAL}$ [2], which is based on *complete* inference algorithms.

Systems based on DLs are hybrid systems which separate the described knowledge in two categories: *terminological* and *assertional* knowledge. The terminological knowledge is generic and refers to classes of objects and their relationships, while the assertional knowledge describes particular instances (individuals).

There are two kinds of terminological knowledge, namely concepts and roles. Concepts are unary predicates interpreted as sets of individuals, while roles represent binary predicates interpreted as binary relations between individuals.

The terminological description language usually provides a variety of concept and role constructors, including the boolean operators (conjunction $\sqcap$, disjunction $\sqcup$, and negation $\neg$). Value- ($\forall R\colon C$), existential- ($\exists R\colon C$) and number restrictions ($\leq_n R$, $=_n R$, $\geq_n R$) as well as enumerations of instances ($\{x_1, \ldots, x_n\}$) are some of the most important concept constructors. We could also mention the following role constructors: $id(C)$ (the restriction of the identity role to the concept C), R^{-1} (role inverse), $R\lfloor C$ (range restriction), $R_1 \circ R_2$ (role composition) and R^* (reflexive-transitive closure).

The *terminological knowledge base* (also called *TBox*) consists of concept and role definitions (which can be necessary, or necessary and sufficient definitions, or even general concept inclusions of the form $C_1 \subset C_2$, with C_1 and C_2 general concept terms). In the following, we shall assume that the terminological axioms are general inclusions, the other concept definitions being easily reducible to inclusions. In particular, we shall allow multiple definitions of concepts and termino-

[1] This research was partially supported by the European Community project *PEKADS* (CP-93-7599).
[2] The $id(C)$ - *reg*ular closure of the $\mathcal{ALC}$ language.
[3] Also known as terminological logics, term subsumption languages or frame-based systems.

ECAI 96. *12th European Conference on Artificial Intelligence*
Edited by W. Wahlster
Published in 1996 by John Wiley & Sons, Ltd.

logical cycles.

Following [1, 7], we can reduce the consistency test of some concept D modulo a given terminology $\mathcal{T}$ to the "pure" consistency test[4] of the concept $D \sqcap \forall \mathcal{R}^*: C_{\mathcal{T}}$, where $\mathcal{R} = \coprod_i R_i$, R_i being the role names occurring in $\mathcal{T}$ and D, while $C_{\mathcal{T}} = \prod_i (\neg A_i \sqcup B_i)$ for $\mathcal{T} = \{A_1 \subset B_1, \ldots, A_n \subset B_n\}$.

The *assertional knowledge base* (also called $ABox$) consists of assertional axioms, i.e. assertions of instances of concepts and roles.

The above-mentioned results allow us to concentrate in the following on the "pure" consistency (satisfiability) test.

2 The terminological language $\mathcal{ALC}^\star$

The terminological language we are using in our knowledge representation system $\mathcal{R}eg\mathcal{AL}$ is $\mathcal{ALC}^\star$. $\mathcal{ALC}^\star$ is the regular closure of the well-known language $\mathcal{ALC}$ of Schmidt-Schauß and Smolka [8] extended with the role constructor $id(C)$.

Besides the concept constructors of $\mathcal{ALC}$ (boolean operators plus existential- and value restrictions), $\mathcal{ALC}^\star$ admits the following role constructors which are interpreted according to the following semantic rules

$$(R_1 \sqcup R_2)^{\mathcal{I}} = R_1^{\mathcal{I}} \cup R_2^{\mathcal{I}} \qquad (R_1 \circ R_2)^{\mathcal{I}} = R_1^{\mathcal{I}} \circ R_2^{\mathcal{I}}$$
$$(R^*)^{\mathcal{I}} = \bigcup_{n \geq 0} (R^{\mathcal{I}})^n \qquad id(C)^{\mathcal{I}} = \{(x,x) | x \in C^{\mathcal{I}}\}.$$

In the following, we shall present *complete* inference algorithms[5] for the terminological language $\mathcal{ALC}^\star$. By taking advantage of the correspondence of $\mathcal{ALC}^\star$ with the propositional dynamic logic (PDL) of programs [7], we shall be able to apply our algorithms for deciding the validity and satisfiability of formulas in PDL too.

2.1 Complete decision algorithms for $\mathcal{ALC}^\star$

The satisfiability (consistency) of a concept in our terminological language can be tested by using a variant of *tableaux calculus* adapted to this specific context. Starting from a formula which implicitly asserts the satisfiability of the given concept, the calculus tries to construct a model of the respective formula. In doing so, it may discover obvious contradictions (clashes) and report the inconsistency of the original formula, or it may come up with a complete clash-free model, thus proving the satisfiability of the formula. This method is directly applicable only if the language possesses the *finite model property* (which is fortunately the case with $\mathcal{ALC}^\star$), since it is obviously impossible to explicitly construct infinite models in a finite amount of time. However, it can also be applied to languages lacking the finite model property whenever finite pseudo-models can be constructed instead of models. Such pseudo-models can then be unwound into potentially infinite models.

The tableaux calculus combines two different processes. The first is analogous to a refutation theorem prover which tries to discover contradictions, while the second concentrates on building models. In [5] a variant of the tableaux calculus (called rule-based calculus operating on constraints) is used for obtaining complete decision procedures for the satisfiability problem in the languages ranging between $\mathcal{ALC}$ and $\mathcal{ALCFNR}$. On the other hand, Franz Baader [1] succeeds in obtaining a practical decision algorithm for the regular closure $\mathcal{ALC}_{reg}$ of $\mathcal{ALC}$. As far as we know, no practical decision algorithms for languages more expressive than $\mathcal{ALC}_{reg}$ are known.

Adding the role constructor $id(C)$ to the language $\mathcal{ALC}_{reg}$ increases the expressivity but introduces substantial complications in the inference algorithms. These complications are mainly due to the fact that existential restrictions are no longer *separable* in the language $\mathcal{ALC}^\star$.

The complete satisfiability checking algorithm is a consequence of the *reduction* and *cycle-characterization* theorems presented below. The idea of the algorithm consists in reducing the satisfiability of a given concept to the satisfiability of several simpler concepts. This reduction process can be alternatively viewed as a process of model construction. In order to ensure the termination of the algorithm, we have to check for the presence of cycles at each reduction step. In case a cycle has been detected, the cycle-characterization theorem is used to determine its nature. As in the case of $\mathcal{ALC}_{reg}$, only the good cycles lead to a model, the bad cycles being merely shorthands for infinite reduction chains.

The satisfiability testing algorithm, presented below, involves a *preprocessing step* in which the following computations are performed:

1) The concept C to be tested is brought to the *negation normal form (nnf)*. The main difference viz. $\mathcal{ALC}_{reg}$ consists in having to consider the concepts I within $id(I)$ roles too. This has to be done depending on the context in which the role $id(I)$ appears (i.e. within an $\forall$ or a $\exists$ restriction) in order to facilitate the extraction of the proper conjuncts of C. More precisely, if $id(I)$ appears in an $\exists$ restriction, then $rnf_\exists(id(C)) = id(nnf(C))$, while if it occurs in an $\forall$ restriction, then $rnf_\forall(id(C)) = id(\neg nnf(\neg C))$.
2) Since comparisons between role expressions R occurring in C are quite frequent (especially when testing the existence of cycles), it seems to be a good idea to bring the roles R to a canonical form. This can be done by constructing for each role R the corresponding deterministic finite automaton DFA and by minimizing the disjoint union of these automata. The initial states of the resulting minimal deterministic finite automaton $mDFA$ represent the canonical forms of the roles occurring in C.

In the following, we shall make no distinction between a role, its corresponding state in the $mDFA$ and the language accepted starting from this state. Also, the following substitutions are performed for all value- and existential restrictions in which $\varepsilon \in R$ (or, equivalently, the state of the $mDFA$ corresponding to R is final):

$$\forall R: Ca \quad \rightarrow \quad Ca \sqcap \forall (R \setminus \{\varepsilon\}): Ca$$
$$\exists R: Ce \quad \rightarrow \quad Ce \sqcup \exists (R \setminus \{\varepsilon\}): Ce.$$

The actual satisfiability testing algorithm extracts a conjunct of the given concept at a time, checks for possible cycles and clashes, then removes the *separable* existential restrictions

[4] modulo the empty terminology

[5] The validity and satisfiability problems in $\mathcal{ALC}^\star$ are known to be decidable (more precisely, EXPTIME–complete).

For reasons of brevity, in presenting the algorithm we shall confine ourselves to the terminological component of the language, i.e. we shall not deal with ABox individuals, nor with enumerations.

and subsequently tries to determine the satisfiability of the remaining *nonseparable* conjunct.

$satisfiable(C)$
  $C' \leftarrow nnf(C)$
  $uDFA \leftarrow \bigcup_{R\ occurs\ in\ C'} role\_to\_DFA(R)$
  $mDFA \leftarrow minimize(uDFA)$
  $C'' \leftarrow roles\_to\_mStates(C')$
  $sat(C'', [])$
□

$sat(C, L)$
  $Conj \leftarrow conjunct(C)$
  $sat\_conjunct(Conj, L)$
□

$sat\_conjunct(Conj, L)$
  if $cycle(Conj, L, \uparrow GoodBad)$ **then**
    if $GoodBad = good$ **then** *succeed*
    else *fail*
  else
    $\overline{Conj} \leftarrow proper\_conjunct(Conj)$
    assign a new unique label Ne *to all* $\exists^{no\_label} Re{:}Ce$ *restrictions*
    // $\overline{Conj} = \prod_i C_i \sqcap \prod_j \exists^{Ne} Re_j{:}Ce_j \sqcap \prod_k \forall \overline{Ra_k}{:}Ca_k$
    if $\prod_i C_i$ *contains a clash (i.e.* $C_{i_1} = \neg C_{i_2}$*)* **then**
      fail
    else
      // solve the separable $\exists$ *restrictions*
      // and collect the nonseparable ones
      $NS\_E \leftarrow sat\_separable\_exists(\prod_j \exists Re_j{:}Ce_j \sqcap \prod_k \forall \overline{Ra_k}{:}Ca_k, [\sqcap\_node(Conj)|L])$
      // solve the nonseparable $\exists$ *restrictions*
      $sat\_nonseparable\_exists(\prod_i C_i \sqcap NS\_E \sqcap \prod_k \forall \overline{Ra_k}{:}Ca_k, [\sqcap\_node(Conj)|L])$
    □
  □
□

$sat\_separable\_exists(\prod_j \exists Re_j{:}Ce_j \sqcap \prod_k \forall \overline{Ra_k}{:}Ca_k, L)$
  $\rightarrow NS\_E$ *// conjunct of nonseparable* $\exists$ *restrictions*
  $NS\_E \leftarrow \top$
  forall $\exists Re{:}Ce$ *in* $\prod_j \exists Re_j{:}Ce_j$
    $sat\_exists(\exists \overline{Re}{:}Ce \sqcap \prod_k \forall \overline{Ra_k}{:}Ca_k, L)$
    or *// nondeterministic choice*
    $NS\_E \leftarrow \exists Re{:}Ce \sqcap NS\_E$
  □
  return NS_E
□

$sat\_exists(\overline{C}_\exists, L)$
  $sat\_exists\_solved(\overline{C}_\exists, L)$
  or *// nondeterministic choice*
  $sat\_exists\_postponed(\overline{C}_\exists, L)$
□

$sat\_exists\_solved(\overline{C}_\exists, L)$
  // $\overline{C}_\exists = \exists \overline{Re}{:}Ce \sqcap \prod_k \forall \overline{Ra_k}{:}Ca_k$
  if *there exists an* $R \in Re$ *such that* $R \neq id(\cdot)$ **then**
    $Ca' \leftarrow \prod_{R \in Ra_k} Ca_k \sqcap \prod_{R^{-1}Ra_k \setminus \{\varepsilon\} \neq \emptyset} \forall (R^{-1}Ra_k \setminus \{\varepsilon\}){:}Ca$
    // solve the $\exists$ *restriction*
    $sat(Ce \sqcap Ca', L)$
  else *fail*
□

$sat\_exists\_postponed(\overline{C}_\exists, L)$
  // $\overline{C}_\exists = \exists^{Ne} \overline{Re}{:}Ce \sqcap \prod_k \forall \overline{Ra_k}{:}Ca_k$
  let $R^{-1}Re$ *be the target state of the transition* $Re \xrightarrow{R} R^{-1}Re$ *with* $R \neq id(\cdot)$
  $Ca' \leftarrow \prod_{R \in Ra_k} Ca_k \sqcap \prod_{R^{-1}Ra_k \setminus \{\varepsilon\} \neq \emptyset} \forall (R^{-1}Ra_k \setminus \{\varepsilon\}){:}Ca$
  // postpone the $\exists$ *restriction*
  $sat(Ca' \sqcap \exists^{Ne}(R^{-1}Re \setminus \{\varepsilon\}){:}Ce, L)$
□

$sat\_nonseparable\_exists(\prod_i C_i \sqcap NS\_E \sqcap \prod_k \forall \overline{Ra_k}{:}Ca_k, L)$
  $C \leftarrow \prod_i C_i \sqcap \prod_k \forall \overline{Ra_k}{:}Ca_k$
  forall $\exists^{Ne} Re{:}Ce$ *in* NS_E
    if $id(I) \in Re$ **then**
      $C \leftarrow (I \sqcap Ce) \sqcap C$
    else *fail*
    or *// nondeterministic choice*
    if $id(I)^{-1}Re \setminus \{\varepsilon\} \neq \emptyset$ **then**
      $C \leftarrow \left[I \sqcap \exists^{Ne}(id(I)^{-1}Re \setminus \{\varepsilon\}){:}Ce\right] \sqcap C$
    else *fail*
  □
  $sat(C, L)$
□

Definition 1 *A restriction* $\exists Re_j{:}Ce_j$ *is called* **separable** *w.r.t. the proper conjunct*[6] $\overline{C}_\sqcap = \prod_i C_i \sqcap \prod_j \exists Re_j{:}Ce_j \sqcap \prod_k \forall \overline{Ra_k}{:}Ca_k$ *iff the concept* $\overline{C}_{\exists_j} = \exists \overline{Re_j}{:}Ce_j \sqcap \prod_k \forall \overline{Ra_k}{:}Ca_k$ *is satisfiable. The proper conjunct* $\overline{C}_\sqcap$ *itself is called* **nonseparable** *iff none of its* $\exists Re_j{:}Ce_j$ *restrictions is separable.*

There are two possibilities of proving the satisfiability of the concept $\overline{C}_{\exists_j}$, namely by *solving* the existential restriction, or by *postponing* it.

Theorem 2 (reduction of the separable existential restrictions) *The concept* $\overline{C}_{\exists_j}$ *above is satisfiable iff one of the following two conditions is met:*
1) there exists a role name $R \in \overline{Re_j}$ *such that* $C_{solved}(R) = Ce_j \sqcap C_\forall(R)$ *is satisfiable, or*
2) there exists a role name R *s.t.* $R^{-1}\overline{Re_j} \setminus \{\varepsilon\} \neq \emptyset$ *and* $C_{postponed}(R) = \exists (R^{-1}Re_j \setminus \{\varepsilon\}){:}Ce_j \sqcap C_\forall(R)$ *is satisfiable, where*

$$C_\forall(R) = \prod_{R \in Ra_k} Ca_k \sqcap \prod_{R^{-1}Ra_k \setminus \{\varepsilon\} \neq \emptyset} \forall (R^{-1}Ra_k \setminus \{\varepsilon\}){:}Ca_k.$$

[6] In $\mathcal{ALC}^\star$, it is important to distinguish between *simple* and *proper* conjuncts. The *simple conjuncts* are the ones obtained by ignoring possible $id(I)$ roles that could occur in the given concept C. The *proper conjuncts* can be obtained from the simple ones by taking into account the implicit disjunctions induced by possible $id(I)$ transitions of roles Ra occurring in value restrictions $\forall Ra{:}Ca$. For instance, $\forall id(I){:}C = \neg I \sqcup C$.

In a similar way, the (nonseparable) existential restrictions from a nonseparable conjunct can be solved or postponed w.r.t. $id(I)$ transitions, but they cannot be separated because of possible interactions between the concepts I.

Theorem 3 (reduction of the nonseparable conjuncts)
A nonseparable proper conjunct $\overline{C}_\sqcap$ *is satisfiable iff at least one of the concepts* $\prod_i C_i \sqcap \prod_j red_{id(I)}(\exists Re_j\!:\!Ce_j) \sqcap \prod_k \forall \overline{Ra}_k\!:\!Ca_k$ *is satisfiable, where* $red_{id(I)}(\exists Re_j\!:\!Ce_j) =$

$$\begin{cases} I \sqcap Ce_j & \text{if } id(I) \in Re_j \\ I \sqcap \exists \left(id(I)^{-1} Re_j \setminus \{\varepsilon\}\right)\!:\!Ce_j & \text{if } id(I)^{-1} Re_j \setminus \{\varepsilon\} \neq \emptyset \end{cases}$$

is the reduction of the restriction $\exists Re_j\!:\!Ce_j$ *w.r.t. the transition* $id(I)$.

In order to be able to determine whether a given existential restriction has been obtained by *postponing* or by *solving* another existential restriction involved in a cycle, we shall attach a unique label N to each existential restriction $\exists^N Re\!:\!Ce$. All existential restrictions are initially unlabeled. An unlabeled restriction $\exists^{no\_label} Re\!:\!Ce$ receives a new unique label N_j only when it reaches the "top level" of a conjunct:[7]

$$\overline{C}_\sqcap = \prod_i C_i \sqcap \prod_j \exists^{N_j} Re_j\!:\!Ce_j \sqcap \prod_k \forall \overline{Ra}_k\!:\!Ca_k.$$

When an existential restriction is postponed, its label is conserved and can be used to track an uninterrupted chain of postponings. Such a chain cannot correspond to a model unless at least one of the existential restrictions in the chain is eventually solved.

In the following, we shall see how the labels can be used to determine the nature of cycles. Let $\overline{C}_\sqcap$ and $\overline{C}'_\sqcap$ be the two concepts involved in a cycle. $\overline{C}_\sqcap$ and $\overline{C}'_\sqcap$ are equal, except maybe the labels N_j and N'_j of the existential restrictions $(j = 1, \ldots, n)$. Such a cycle will be represented by the *label-correspondence table* $\begin{pmatrix} N_1 & N_2 & \ldots & N_n \\ N'_1 & N'_2 & \ldots & N'_n \end{pmatrix}$, each column of this table being related to equal existential restrictions $\exists^{N_i} Re\!:\!Ce = \exists^{N'_i} Re\!:\!Ce$ from $\overline{C}_\sqcap$ and $\overline{C}'_\sqcap$ respectively. Because $\exists$ restrictions get unique labels when they reach the top level of a conjunct, we have $N_i \neq N_j$ and $N'_i \neq N'_j$ for $i \neq j$. The following theorem can be used in determining the nature of a cycle.

Theorem 4 (cycle characterization)
A cycle represented by the label correspondence table above is **bad** *(i.e. it does not induce a model) iff the label correspondence table contains a cyclic permutation, i.e. there exists a subset of indices* $\{j_1, j_2, \ldots, j_k\} \subset \{1, \ldots, n\}$ *such that* $N_{j_1} = N'_{j_2}$, $N_{j_2} = N'_{j_3}$, $\ldots$, $N_{j_{k-1}} = N'_{j_k}$, $N_{j_k} = N'_{j_1}$.

3 Representing epistemic operators in description logics

Since we are aiming at a unified architecture for knowledge representation based on description logics, we shall show that DLs are powerful enough to represent epistemic operators corresponding to the majority of modal logics of knowledge and belief.

In modal logic, an agent can imagine a set of possible worlds linked with the real world by the *accessibility relation*. The facts p known by the agent are facts which are true in all possible worlds.

Modal formulas are constructed by using the usual logical connectives together with the modal operators $\Box$ (necessity) and $\Diamond$ (possibility). The necessity modal operator $\Box$ will be interpreted in the following as an epistemic operator, the formula $\Box_i p$ being understood as "the agent i knows the fact p".

Because of the fact that there is no unique interpretation of the modal notions of "necessity", "possibility", "knowledge", "belief" etc., there exists a large variety of modal systems which can be distinguished by the properties of the accessibility relation. Imposing, for instance, the reflexivity of the accessibility relation ρ in the modal system **T** is equivalent to requiring the truth of knowledge, while imposing the seriality of ρ leads to the consistency of knowledge.

The *multi-modal logics of knowledge* **K**, **T**, **S4**, **S5**, **B** can be embedded in a description logic by using the following satisfiability preserving translations of the modal (epistemic) operators:

$$\begin{aligned} \Box_i p &\longmapsto \forall \mathcal{L}(R_i)\!:\!p' \\ \Diamond_i p &\longmapsto \exists \mathcal{L}(R_i)\!:\!p'. \end{aligned}$$

In this way, problems formulated in terms of modal operators can be reduced to problems in a DL which can be solved using the inference algorithms from the preceding sections. Here $\mathcal{L}$ denotes the particular modal system, while R_i is an arbitrary role name representing the agent i. The role $\mathcal{L}(R_i)$ stands for the accessibility relation and possesses all the properties this relation should have in the system $\mathcal{L}$. Thus, we could read the formula $\forall \mathcal{L}(R_i)\!:\!p'$ as: "the agent i knows the fact p' w.r.t. the modal system $\mathcal{L}$" ($\mathcal{L}$ gives us here the *type* of knowledge).

Modal logic of knowledge $\mathcal{L}$	Axioms	$\mathcal{L}(R)$
K.	K	R
T.	KT	$R \cup id$
S4.	KT4	R^*
S5.	KT45	$(R \cup R^{-1})^*$
B.	KTB	$R \cup id \cup R^{-1}$

Table 1. The accessibility relation $\mathcal{L}(R)$ in the modal logics of knowledge

Table 1 presents the expression of $\mathcal{L}(R)$ for some of the most important *modal logics of knowledge*. Note that all these formulas for $\mathcal{L}(R_i)$ are quite intuitive if we interpret the role R_i as a one-step access to the state of knowledge of the agent i. Consider for instance the case of the modal logic $\mathbf{S4} = KT4$, in which the axiom T asserts the truth of knowledge, while the axiom 4 requires the property of *positive introspection*. In this case, the expression of the accessibility relation $S4(R_i) = R_i^*$ shows that the agent i can access knowledge about knowledge

[7] This happens in *sat_conjunct* after extracting a proper conjunct from a simple one.

(during positive introspection) in multiple accesses using the one-step access action R_i.

In the weakened versions $\mathcal{OL}$ and $\mathcal{OL}^+$ of the systems $\mathcal{L}$ above we have to replace the axioms of reflexivity T and symmetry B with the weaker versions corresponding to *almost reflexivity* $\Box(\Box p \rightarrow p)$ and *almost symmetry* $\Box(\Diamond\Box p \rightarrow p)$ respectively. Such logics, in which the truth of knowledge is replaced by the requirement that beliefs should be believed to be true, are appropriate as *modal logics of belief*. The expression of the accessibility relation in these logics takes the form

$$\mathcal{OL}^+(R_i) = \mathcal{OL}(R_i) = \mathcal{L}(R_i) \circ id(q_i),$$

where q_i are atomic formulas. In the case of the "deontic" systems $\mathcal{OL}^+$, in which the beliefs are required to be consistent, we also have to assert the facts $\exists\mathcal{OL}^+(R_i)\colon \top$ (or, equivalently, $\exists\mathcal{L}(R_i)\colon q_i$).

The common knowledge and common belief operators take the form $\mathcal{C} = [(\coprod_i \mathcal{L}(R_i))^+]$ and $\mathcal{D} = [(\coprod_i \mathcal{OL}(R_i))^+]$ respectively.

The main advantage of our unifying approach is that the various types of knowledge corresponding to the aforementioned modal systems can be *amalgamated* in a single system. Not only is it possible to describe in $\mathcal{RegAL}$ the knowledge/beliefs of several agents, but the different agents could have different epistemic operators with distinct modal properties so that we could study, for example, the interaction between an agent whose *knowledge* is necessarily true and another agent whose *beliefs* are just *consistent* and *believed to be true*, but not necessarily true in reality. One could even have more than one epistemic operator attached to the same agent in order to distinguish its beliefs from its knowledge. Of course, in $\mathcal{RegAL}$ epistemic operators can be nested in an unrestricted fashion and they could even mention actions and plans. Also, the actions of some agent could modify the knowledge or beliefs of another agent so that it becomes possible to study the *communication* between agents in a unified framework.

Our method of integrating epistemic operators in a DL is much simpler than other approaches which, on one hand, can usually deal with only one single type of knowledge at a time and, on the other, had to develop special purpose algorithms for treating the epistemic operators (since the underlying DL has usually a too low expressivity to be able to express epistemic operators directly).

4 Concluding remarks

This paper tries to present a unified approach to the domain of knowledge representation from the viewpoint of description logics. We have shown that DLs are powerful enough to represent not only the domain knowledge in a particular application, but also the epistemic operators actions and plans [2] of a set of interacting agents. Because of our unifying approach, all these types of knowledge can be combined in an unrestricted fashion.

In order to support the reasoning involved, we have chosen a decidable terminological language, $\mathcal{ALC}^\star$, for which we have developed "practical" inference algorithms. It should not be surprising that these algorithms are quite complex, because the underlying language has a high expressivity.

Note that although several different algorithms for reasoning in PDL-like logics have been put forward (for example [9]), we argue that most of them, although theoretically optimal, are not good enough if we are aiming at an operational system. While such a system should not have a higher computational complexity than the theoretical worst-case complexity of the decision problem, it should also behave much better in the average (simpler) cases (our "practical" algorithm behaves well in such average cases). But this is not the case with most of the existing algorithms which start by constructing a structure with an exponential number of states by creating all the subsets of the Fischer-Ladner closure of the input formula. (An exception is the paper [6] which uses a "bottom-up" approach, constructing only as much of the model of the input formula as is needed).

Although developed independently, our algorithm is similar in spirit, but not in realization, to the above-mentioned algorithm of Pratt. One of the improvements of the present algorithm consists in precomputing the canonical forms of roles occurring in the tested formula (by constructing a minimal DFA, like in [1]) which simplifies the cycle testing that has to be done for ensuring termination.

Also, the fact that the above-mentioned algorithm expresses formula conjunctions by means of tests $p \wedge q = \langle p? \rangle q$, leads to inefficient reasoning at the level of the boolean connectors. Our method allows, on the other hand, a more refined inference control, at least at the level of the boolean operators.

The resulting system, called $\mathcal{RegAL}$, is implemented in PROLOG and will be used in a bigger knowledge-based systems development environment.

Note that the language $\mathcal{CATS}$ [4] is a similar very expressive PDL-like description logic. However, while the paper [4] uses a translation approach that relies on existing decision algorithms for PDL, our approach stresses the importance of improving and extending the existing algorithms for PDL in order to obtain better average-case performances.

REFERENCES

[1] BAADER F. *Augmenting concept languages by the transitive closure: An alternative to terminological cycles.* IJCAI-91, pp. 446-451.

[2] BADEA LIVIU. *A unitary theory and architecture for knowledge representation and reasoning* (in Romanian) Research Report A11-2, ICI, 1995.

[3] BRACHMAN R.J., SCHMOLZE J.G. *An Overview of the KL-ONE Knowledge Representation System.* Cognitive Science 9 (2) 1985.

[4] DE GIACOMO G., LENZERINI M. *What's in an Aggregate: Foundations for Description Logics with Tuples and Sets.* IJCAI-95, pp. 801-807.

[5] HOLLUNDER B., NUTT W., SCHMIDT-SCHAUSS M. *Subsumption Algorithms for Concept Description Languages.* ECAI-90, pp. 384-353, Pitman, 1990.

[6] PRATT V.R. *A near-optimal method of reasoning about action.* J. Comp. System Sci. 20 (2) (1980), pp. 231-254.

[7] SCHILD KLAUS. *A correspondence theory for terminological logics: preliminary report.* IJCAI-91, pp. 466-471.

[8] SCHMIDT-SCHAUSS M., SMOLKA G. *Attributive concept descriptions with complements.* AI Jour. 48 (1), pp. 1-26, 1991.

[9] VARDI M.Y., WOLPER P. *Automata-theoretic techniques for modal logics of programs.* J. Comp. System Sci. 32 (1986), pp. 183-221.