Towards a unified architecture for knowledge representation and reasoning based on terminological logics *

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 and reasoning based on terminological (description) logics. The novelty of our approach consists in trying to use description logics not only for representing domain knowledge, but also for describing beliefs, epistemic operators and actions of intelligent agents in an unitary framework. For this purpose, we have chosen a decidable terminological language, called $\mathcal{ALC}_{reg\,+\,id(C)}$, 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 $\mathcal{ALC}_{reg\,+\,id(C)}$ which lies at the heart of our $\mathcal{R}eg\mathcal{AL}$ [1] 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 [10].

1 Term subsumption languages

Term subsumption languages [2] (TSLs) are descendants of the famous KL-ONE language [4] and can be viewed as formalizations of the frame-based knowledge representation systems.

The relationship between TSLs and logic is analogous to the relationship between structured and unstructured programming languages. Indeed, the TSLs 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, TSLs 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 TSLs may possess *decidable* inference problems while retaining a fairly high expressivity which enables them to represent complex ontologies.

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 ($\leq_n R$, $=_n R$, $\geq_n R$), role-value maps ($R_1 \sqsubset R_2$) and structural descriptions ($C : R$) 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), R^* (reflexive-transitive closure) and $R_1 \prec R_2$ (bindings used in structural descriptions).

Not all of the above constructors are independent. For example, role-value maps and structural descriptions can be expressed in a language that admits role negation [3] as:

$$\begin{aligned} R_1 \sqsubset R_2 &= \forall(R_1 \sqcap \neg R_2)\colon \bot \\ C : R &= \exists R\colon C, \quad \textit{where} \\ R &= R_1 \prec Q_1 \sqcap \ldots \sqcap R_n \prec Q_n \\ R_i \prec Q_i &= \neg(R_i \circ \neg Q_i^{-1}). \end{aligned}$$

Role-value maps and structural descriptions usually lead to very expressive but *undecidable* languages [12, 8]. These observations suggest the following conjecture: *"The only cause of undecidability of a reasonably expressive terminological language is the irreducible presence of role negation in the language."* Note that concept negation is usually harmless w.r.t. decidability, as opposed to role negation which usually leads to undecidable languages [9].

*This research was partially supported by the European Community project $^{PE}KADS$ (CP-93-7599).

[1] The $id(C)$ - *Reg*ular closure of the $\mathcal{ALC}$ language.

[2] Also known as terminological (or description) logics.

[3] and also other common concept and role constructors

2 Complete decision algorithms for the terminological language $\mathcal{ALC}_{reg\,+\,id(C)}$

The terminological language we are using in our knowledge representation system $\mathcal{R}eg\mathcal{AL}$ is $\mathcal{ALC}_{reg\,+\,id(C)}$, the regular closure of the well-known language $\mathcal{ALC}$ of Schmidt-Schauß and Smolka [11] extended with the role constructor $id(C)$.

In the following, we shall present *complete* inference algorithms [4] for $\mathcal{ALC}_{reg\,+\,id(C)}$. By taking advantage of the correspondence of $\mathcal{ALC}_{reg\,+\,id(C)}$ with the propositional dynamic logic (PDL) of programs [10], we shall be able to apply our algorithms for deciding the validity and satisfiability of formulas in PDL too.

As far as we know, there exists a single TSL system with complete inference algorithms and a reasonably high expressivity, namely $\mathcal{KRIS}$ [2]. The terminological language $\mathcal{ALCFNR}$ provided by $\mathcal{KRIS}$ extends the standard language $\mathcal{ALC}$ with attributes (functional roles), number restrictions and role conjunctions.

The language $\mathcal{ALC}_{reg\,+\,id(C)}$ we are using in $\mathcal{R}eg\mathcal{AL}$ was chosen having somewhat different goals in mind, namely to be able to represent procedural knowledge, actions and epistemic operators in our descriptive logic. Number restrictions and role conjunctions wouldn't have been very helpful in this context.

The satisfiability (consistency) of a concept in our terminological language can be tested by using a variant of the well known *tableaux calculus*, adapted to this specific context [6]. 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 $\mathcal{ALC}_{reg\,+\,id(C)}$).

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 [6] 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 $\mathcal{ALC}_{reg\,+\,id(C)}$.

The complete satisfiability checking algorithm is a consequence of the *reduction* and *cycle-characterization* theorems presented in [3]. 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 in figure 1, 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 $rnf_{\exists}(id(C)) = id(nnf(C))$, and 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.
3) Finally, the procedure *roles_to_mStates* replaces the roles occurring in C with the corresponding states of the *mDFA*. The replacements affect the concepts I inside $id(I)$ transitions of the *mDFA* too.

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):

$$\begin{aligned} \forall R\colon Ca &\rightarrow Ca \sqcap \forall(R \setminus \{\varepsilon\})\colon Ca \\ \exists R\colon Ce &\rightarrow Ce \sqcup \exists(R \setminus \{\varepsilon\})\colon Ce. \end{aligned}$$

The actual satisfiability testing algorithm extracts a conjunct of the given concept at a time, removes the *separable* existential restrictions and subsequently tries to determine the satisfiability of the remaining *nonseparable* conjunct.

[4] The validity and satisfiability problems in $\mathcal{ALC}_{reg\,+\,id(C)}$ are known to be decidable (more precisely, EXPTIME-complete).

$satisfiable(C)$
  $C' \leftarrow nnf(C)$
  $uDFA \leftarrow \emptyset$
  forall *roles* R *occurring in* C'
    $DFA \leftarrow role\_to\_DFA(R)$
    $uDFA \leftarrow DFA \cup uDFA$
  □
  $mDFA \leftarrow minimize(uDFA)$
  $C'' \leftarrow roles\_to\_mStates(C')$
  $sat(C'', [])$
□

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

$sat\_conjunct(Conj, L)$
  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*
      $sat\_nonseparable\_exists(\prod_i C_i \sqcap NS\_E$
        $\sqcap \prod_k \forall \overline{Ra}_k{:}\, Ca_k, [\sqcap\_node(Conj)|L])$
    □
  □
□

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

Figure 1: The satisfiability testing algorithm for concepts in $\mathcal{ALC}_{reg\,+\,id(C)}$

$sat\_separable\_exists(\prod_j \exists Re_j{:}\, Ce_j \sqcap \prod_k \forall \overline{Ra}_k{:}\, Ca_k, L) \rightarrow NS\_E$
  // 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
□

$sat\_nonseparable\_exists(\prod_i C_i \sqcap NS\_E \sqcap \prod_k \forall \overline{Ra}_k{:}\, Ca_k, L)$
  $C \leftarrow \prod_i C_i \sqcap \prod_k \forall \overline{Ra}_k{:}\, Ca_k$
  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*
  □
  $sat(C, L)$
□

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

Definition 1 *A restriction* $\exists Re_j : Ce_j$ *is called* **separable** *w.r.t. the proper conjunct* [5] $\overline{C}_\sqcap = \prod_i C_i \sqcap \prod_j \exists Re_j : Ce_j \sqcap \prod_k \forall \overline{Ra}_k : Ca_k$ *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.

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.

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

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 1 *(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* $N_{j_1} = N'_{j_2}$, $N_{j_2} = N'_{j_3}$, $\ldots$, $N_{j_{k-1}} = N'_{j_k}$, $N_{j_k} = N'_{j_1}$.

[5] In $\mathcal{ALC}_{reg\,+\,id(C)}$, 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, $\forall id(I) : C = \neg I \sqcup C$.

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

3 Representing epistemic operators in terminological logics

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

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 $\Box p$ being understood as "the agent 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 table 1 presents some of the most common modal axioms together with the properties of the accessibility relation they induce.

The most common modal systems are defined by combinations of the modal axioms from table 1. They can be embedded in a term subsumption language by using *satisfiability preserving translations* into the TSL (see also [13]). In this way, problems formulated in terms of (modal) epistemic operators can be reduced to problems in a TSL which can be solved using the inference algorithms from the preceding sections.

The general translation scheme from a modal system

Name	Modal axiom	Property of the accessibility relation	Comments
K.	$\Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)$	valid in every standard Kripke frame	*Kripke's* axiom (*normality* axiom)
D.	$\Diamond\top$	serial	*deontic* axiom
T.	$\Box p \rightarrow p$	reflexive	*knowledge* axiom
B.	$p \rightarrow \Box\Diamond p$ $p \rightarrow \Box\neg\Box\neg p$	symmetric	*Brouwer* axiom
4.	$\Box p \rightarrow \Box\Box p$	transitive	*positive introspection* axiom
5.	$\Diamond p \rightarrow \Box\Diamond p$ $\neg\Box p \rightarrow \Box\neg\Box p$	euclidian	*negative introspection* axiom
U.	$\Box(\Box p \rightarrow p)$	almost reflexive	beliefs are believed to be true
(A.)	$\Box(\Diamond\Box p \rightarrow p)$	almost symmetric	

Table 1: Major modal axioms

into a TSL is the following (p' is the TSL concept corresponding to the modal formula p):

$$\begin{array}{rcl} p & \longmapsto & p \ (\textit{for atomic formulas}) \\ \neg p & \longmapsto & \neg p' \\ p \wedge q & \longmapsto & p' \sqcap q' \\ p \vee q & \longmapsto & p' \sqcup q' \\ \Box p & \longmapsto & \forall \mathcal{L}(R){:}\, p' \\ \Diamond p & \longmapsto & \exists \mathcal{L}(R){:}\, p'. \end{array}$$

Note that the modal operators $\Box$ and $\Diamond$ are translated into value- and existential restrictions in which roles of the form $\mathcal{L}(R)$ occur. Here $\mathcal{L}$ is the particular modal system and R an arbitrary role name representing the agent. The role $\mathcal{L}(R)$ 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 $\forall\mathcal{L}(R){:}\, p'$ as: "the agent R knows the fact p' w.r.t. the modal system $\mathcal{L}$" ($\mathcal{L}$ gives us here the *type* of knowledge).

The table 2 presents the expression of $\mathcal{L}(R)$ for the most important *modal logics of knowledge* (in which knowledge is required to be true [7]) while the table 3 does the same thing for the *modal logics of belief* (in which beliefs are believed to be true). The axioms of reflexivity **T** and symmetry **B** from the modal logics of knowledge are replaced in the modal logics of belief by the weaker versions **U** (almost reflexivity) and **A** (almost symmetry) respectively.

Adding the deontic axioms $\exists\mathcal{OL}^+(R){:}\,\top$ or, equivalently, $\exists\mathcal{L}(R){:}\,q$ to the systems $\mathcal{OL}(R)$ in table 3 leads to the *deontic systems* $\mathcal{OL}^+(R)$ in which the beliefs are required to be *consistent*. Note that

$$\mathcal{OL}^+(R) = \mathcal{OL}(R) = \mathcal{L}(R) \circ id(q).$$

[7] except perhaps in the system **K**.

System	Axioms	$\mathcal{L}(R)$
K.	K	R
T.	KT	$R \sqcup id$
S4.	KT4	R^*
S5.	KT5	$(R \sqcup R^{-1})^*$
B.	KTB	$R \sqcup id \sqcup R^{-1}$

Table 2: The accessibility relation $\mathcal{L}(R)$ in the modal logics of knowledge

System	Axioms	$\mathcal{OL}(R)/\mathcal{OL}^+(R)$
OK/OK$^+$.	K/KD	$R \circ id(q)$
OT/OT$^+$.	KU/KDU	$(R \sqcup id) \circ id(q)$
OS4/OS4$^+$.	K4U/KD4U	$R^* \circ id(q)$
OS5/OS5$^+$.	K45/KD45	$(R \sqcup R^{-1})^* \circ id(q)$
OB/OB$^+$.	KUA/KDUA	$(R \sqcup id \sqcup R^{-1}) \circ id(q)$

Table 3: The accessibility relations $\mathcal{OL}(R)/\mathcal{OL}^+(R)$ in the modal logics of belief

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. For example, we could describe a multi-agent system in which the knowledge $\mathcal{K}_i$ and beliefs $\mathcal{B}_i$ of the agents i can be mixed in an unrestricted fashion. By attaching a unique role name R_i to each agent i, we can write the epistemic operators corresponding to the knowledge and belief of agent i in the following way [8]

$$\mathcal{K}_i = [\mathcal{R}_i] = [S4(R_i)] = [R_i^*]$$
$$\mathcal{B}_i = [\mathcal{T}_i] = [KD4U(R_i)] = [R_i^* \circ id(q_i)].$$

[8] In order to simplify the notation, we shall write, in the following, $[R]C$ instead of $\forall R{:}\, C$.

where $\mathcal{T}_i$ verifies the deontic axiom $\exists\mathcal{T}_i\colon\top$, or equivalently, $\exists R_i^*\colon q_i$.

The common knowledge and common belief operators are $\mathcal{C} = [(\coprod_i \mathcal{R}_i)^+]$ and $\mathcal{D} = [(\coprod_i \mathcal{T}_i)^+]$ respectively.

Our method of integrating epistemic operators in a TSL is much simpler and more natural than other approaches [5, 7] which, on one hand, could 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 (because the underlying TSL had a too low expressivity to be able to express epistemic operators directly).

4 Representing actions and plans in a TSL

TSLs can be used not only for representing the domain knowledge or epistemic operators, but also for describing actions and plans. In order to develop a theory of action in TSLs, we shall regard a role of a TSL as an action which transforms the states x from the extension of the role's domain into the states y from the extension of its range. Thus, the value restriction $\forall R\colon C$ can be interpreted as the necessary precondition for the action R to achieve the postcondition C.

Conditions/facts from our theory of action will be represented in a TSL by concepts, while actions will be denoted by roles. An action $A : \langle In|Ctx|Out\rangle$ (having In as deleted preconditions, Ctx as context (preserved preconditions) and Out as created postconditions) can be described by the following terminological axiom, which is similar to a *total correctness assertion* from dynamic logic [9]

$$In \sqcap Ctx \subset \forall\exists A\colon (\neg In \sqcap Ctx \sqcap Out)$$

where $\forall\exists R\colon C \stackrel{def}{=} \exists R\colon\top \sqcap \forall R\colon C = \exists R\colon C \sqcap \forall R\colon C$.

The *planning problem* can be stated in the following way: "Given an initial state represented by the concept *Initial*, a final state (goal) *Final* and a repertory of actions $\{A_1, A_2, \ldots, A_n\}$, find a role chain $Plan = A_{i_1} \circ A_{i_2} \circ \ldots \circ A_{i_k}$ (or, more generally, a role term *Plan* formed from the roles $A_1, \ldots, A_n$ by applying the role constructors) such that $Initial \subset \forall\exists Plan\colon Final$."

This last equation assures us that the compound action *Plan* is applicable in a state verifying the preconditions *Initial* and that its application will produce a state verifying the goals *Final*.

5 Conclusions

This paper tries to present a unified approach to the domains of knowledge representation and reasoning from the viewpoint of terminological (description) logics. We have shown that TSLs are powerful enough to represent not only the domain knowledge in a particular application, but also the epistemic operators, actions and plans 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, $\mathcal{ALC}_{reg\,+\,id(C)}$, for which we have developed the key inference algorithms. It should not be surprising that these algorithms are quite complex, because the underlying language has a high expressivity.

The resulting system, called $\mathcal{R}eg\mathcal{AL}$, is implemented in PROLOG and will be used in a very powerful knowledge-based systems development environment.

[9]This similarity should not be surprising since the planning problem is similar to the problem of program synthesis starting from input/output specifications.

References

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

[2] BAADER F., HOLLUNDER B. *KRIS: Knowledge Representation and Inference System – System Description.* DFKI TM-90-03.

[3] BADEA LIVIU. *A unitary theory and architecture for knowledge representation and reasoning in Artificial Intelligence* (in Romanian) PhD thesis, Bucharest Polytechnic University, 1994.

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

[5] DONINI F.M., LENZERINI M., NARDI D., SCHAERF A., NUTT W. *Adding Epistemic Operators to Concept Languages.* Proceedings KR-92, Boston.

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

[7] LAUX A. *Integrating a Modal Logic of Knowledge into Terminological Logics.* DFKI RR-92-56.

[8] PATEL-SCHNEIDER P.F. *Undecidability of Subsumption in NIKL.* Artificial Intelligence 39 (1989), pp. 263-272.

[9] SCHILD KLAUS. *Undecidability of Subsumption in U.* KIT Report, Technische Universität Berlin, October 1988.

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

[11] SCHMIDT-SCHAUSS M., SMOLKA G. *Attributive concept descriptions with complements.* Artificial Intelligence 48 (1), pp. 1-26, 1991.

[12] SCHMIDT-SCHAUSS M. *Subsumption in KL-ONE is undecidable.* Proceedings KR-89, pp. 421-431.

[13] TUOMINEN H. *Translations from Epistemic into Dynamic Logic.* ECAI-88, pp. 586-588.