Adapting Optimization Techniques to Description Logics with Concrete Domains

Anni-Yasmin Turhan and Volker Haarslev
Department of Computer Science, University of Hamburg, Germany
Email: {turhan, haarslev}@kogs.informatik.uni-hamburg.de

Abstract

In this paper, we demonstrate that the main standard optimization techniques dependency directed backtracking and model merging can be adapted to description logics with concrete domains. We propose algorithms for these techniques for the logics $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCRP}(\mathcal{D})$. Important results of this study are (1) a new requirement for concrete domains in order to enable dependency directed backtracking for all clash types of description logics with concrete domains, and (2) the flat and deep model merging techniques can be fully adapted to $\mathcal{ALC}(\mathcal{D})$ but their applicability to the logic $\mathcal{ALCRP}(\mathcal{D})$ is limited.

1 Motivation

In recent years dramatic advancements in developing new optimization techniques for expressive description logics (DLs) have been achieved. The requirements derived from practical applications of DLs ask for more expressive languages. It is well-known that reasoning about objects from other domains (so-called concrete domains, e.g. the real numbers [1]) is very important for practical applications as well. However, it was unknown whether the new optimization techniques can be adapted to DLs with concrete domains. This paper reports on the first study [7] which analyzes this problem and proposes adapted algorithms for two important optimization techniques, dependency directed backtracking and model merging [4],[3], in the context of TBox reasoning for $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCRP}(\mathcal{D})$. We assume that the reader is familiar with $\mathcal{ALC}(\mathcal{D})$, $\mathcal{ALCRP}(\mathcal{D})$ and their calculi (see [1], [6] and [2] respectively).

1.1 The Description Logics $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCRP}(\mathcal{D})$

In this section we briefly introduce the logics $\mathcal{ALC}(\mathcal{D})$ and its extension $\mathcal{ALCRP}(\mathcal{D})$. Both DLs can be parameterized with an admissible concrete domain $\mathcal{D}$. A concrete domain (CD) consists of a domain and a set of predicates. A CD is

	Syntax	Semantics
$\mathcal{ALC}(\mathcal{D})$	$\exists u_1, \ldots, u_n.P$	$\{a \in \Delta_\mathcal{I} \mid x_1, \ldots, x_n \in \Delta_\mathcal{D}:$ $(a, x_1) \in u_1^\mathcal{I}, \ldots, (a, x_n) \in u_n^\mathcal{I}, (x_1, \ldots, x_n) \in P^\mathcal{D}\}$
$\mathcal{ALCRP}(\mathcal{D})$	$(\exists(u_1, \ldots, u_n)$ $(v_1, \ldots, v_m).Q)$	$\{(a, b) \in \Delta_\mathcal{I} \times \Delta_\mathcal{I} \mid \exists x_1, \ldots, x_n, y_1, \ldots, y_m \in \Delta_\mathcal{D}:$ $(a, x_1) \in u_1^\mathcal{I}, \ldots, (a, x_n) \in u_n^\mathcal{I},$ $(b, y_1) \in v_1^\mathcal{I}, \ldots, (b, y_m) \in v_m^\mathcal{I},$ $(x_1, \ldots, x_n, y_1, \ldots, y_m) \in Q^\mathcal{D}\}$

Throughout this document we use this naming convention:

C,D : concept terms R : primitive or complex role P,Q : predicates f : feature
u_i,v_i : feature chains a,b : abstract individuals x,y : concrete individuals

Table 1: Predicate exists restriction and Role-forming predicate exists restriction

called admissible if (1) the set of predicates for the elements of the domain is closed under negation, (2) the set of predicates contains a predicate for domain membership and (3) the satisfiability problem for conjunctions of predicates is decidable.

The DL $\mathcal{ALC}(\mathcal{D})$ extends the standard logic $\mathcal{ALC}$ with features and so-called *predicate exists restrictions*. The latter ones are concept terms, which allow to refer to objects of the concrete domain by feature chains and to state that a predicate from the concrete domain holds between these objects ($\exists u_1, \ldots, u_n.P$).

The DL $\mathcal{ALCRP}(\mathcal{D})$ in turn extends $\mathcal{ALC}(\mathcal{D})$ with a so-called *role-forming predicate exists restriction*. It allows one to specify roles based on role terms using concrete domain predicates (e.g. $R_c \doteq \exists(u_1, \ldots, u_n)(v_1, \ldots, v_m).P$). Such a *complex role* R_c has some feature chains (the u_i's) starting from the role predecessor a and some feature chains (the v_i's) starting from the successor b of the complex role. All of these chains are referencing concrete domain objects and if the predicate P holds between these objects, the complex role R_c holds between a and b. Complex roles can be used in value and exists restrictions in $\mathcal{ALCRP}(\mathcal{D})$ concept terms. Whenever we mention $\mathcal{ALCRP}(\mathcal{D})$ in this paper we always assume its syntax restricted version (see [2]). For the unrestricted concept terms satisfiability is not decidable. The interpretation of $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCRP}(\mathcal{D})$ concept terms (and role terms) requires two disjoint domains: the abstract domain $\Delta_\mathcal{I}$ and the concrete domain $\Delta_\mathcal{D}$. Due to lack of space we omit a further introduction to the semantics and to concrete domains and refer to [1, 2] for details. The syntax and semantics of the new language elements are shown in Table 1.

In order to determine if D subsumes C the concept term $(C \sqcap \neg D)$ is tested for unsatisfiability. First the term is unfolded with respect to the TBox.[1] In

[1]For DLs with concrete domains concept satisfiability is decidable only for unfoldable TBoxes. Therefore, we consider only terminologies without cyclic or multiple definitions or general concept inclusions.

case of $\mathcal{ALCRP}(\mathcal{D})$ also the complex roles need to be unfolded. Then the term is transformed into negation normal form, resulting in an equivalent term with negation only appearing in front of concept names. It is well-known that the satisfiability of concept terms can be reduced to ABox consistency.

Def. 1 (ABox) *An ABox $\mathcal{A}$ is a finite set of assertional axioms. The following expressions are assertional axioms:* $a : C$ (Concept assertion), $(a, b) : R$ (Role assertion), $(a, b) : f$, $(a, x) : f$ (Feature assertions), $(x_1, \ldots, x_n) : P$ (Predicate assertion).

$\mathcal{ALC}(\mathcal{D})$ ABoxes contain only role assertions for primitive roles, while the ABoxes for $\mathcal{ALCRP}(\mathcal{D})$ also allow role assertions for complex roles.

So in order to test if C is satisfiable, the ABox $\mathcal{A} = \{a : C\}$ is checked for consistency with the tableaux calculus. We only do sketch some properties of the tableaux calculi here. The *trace technique* is a control structure for tableaux provers, which allows them to discard parts of a model during a proof, by exploiting that models of some DLs can be represented as trees and therefore partitioned into "sub-tableaux". Currently many DLs for which optimization techniques have been applied use a partitioning allowing for sub-tableaux with a single individual. In contrast to this $\mathcal{ALC}(\mathcal{D})$ sub-tableaux have to be be partioned into feature-complete sub-tableaux in order to use the trace technique (see [6]). A *feature-complete sub-tableau* contains all individuals that are connected via a feature-chain (with possibly a single feature) starting from the root individual of the sub-tableau and it contains their concept, feature and predicate assertions.

For $\mathcal{ALCRP}(\mathcal{D})$ models the tree model property does not hold due to the *phantom edges* which are implicit role-filler relationships established by the use of value restrictions for complex roles (see [2] for details). Some of the tableaux rules have properties needed for the application of optimization techniques: *Non-deterministic rules* are tableaux rules that yield more than one successor ABox. Rules with *required assertions* have preconditions which require assertions to be present in the ABox in addition to the assertion which is expanded by the rule. *Generating rules* are tableaux rules, which generate new individuals in the ABox. The non-deterministic rules and the required assertions are needed for dependency directed backtracking, while the generating rules are important for the model merging technique. These properties of the completion rules of $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCRP}(\mathcal{D})$ are recapitulated in Table 2.

	non-deterministic	required assertions	generating
$\mathcal{ALC}(\mathcal{D})$	R$\sqcup$	R$\forall$C	R$\exists$C, R$\exists$P
$\mathcal{ALCRP}(\mathcal{D})$	R$\sqcup$, RChoose	R$\forall$C, RChoose	R$\exists$C, R$\exists$P, Rr$\exists$P

Table 2: Properties of the tableaux rules

The assertions in the ABox are expanded until no tableaux rule is applicable or a clash occurs and no more alternatives are to be explored. In DLs with CDs we have the following clash types:

Def. 2 (Clash types, culprit) *An ABox is called* contradictory *if any of the following* culprit sets *is a subset of $\mathcal{A}$:*

- **primitive Clash** $\{a : C, \quad a : \neg C\} \subseteq \mathcal{A}$
- **Feature Domain Clash** $\{(a, x) : f, \quad (a, b) : f\} \subseteq \mathcal{A}$
- **All Domain Clash** $\{(a, x) : f, \quad a : \forall f.C\} \subseteq \mathcal{A}$
- **Concrete Domain Clash**
  $\{(x_1^{(1)}, \ldots, x_{n_1}^{(1)}) : P_1, \ \ldots, \ (x_1^{(k)}, \ldots, x_{n_k}^{(k)}) : P_k\} \subseteq \mathcal{A}$ *and the conjunction* $\bigwedge_{i=1}^{k} P_i(x^{(i)})$ *is not satisfiable in* $\mathcal{D}$.

A culprit *is an assertion that is an element of a culprit set.*

A test for a clash type is performed after every application of a completion rule that might generate a clash of that type. An ABox $\mathcal{A}$ contains a *fork* iff either: $\{(a, b) : f, (a, c) : f\} \subseteq \mathcal{A}$ or $\{(a, x) : f, (a, y) : f\} \subseteq \mathcal{A}$. A fork is eliminated by replacing all occurrences of b in $\mathcal{A}$ with c (x with y resp.). Tests for forks are carried out after application of a generating rule which created a feature successor.

Def. 3 (complete ABox) *A fork-free ABox $\mathcal{A}$ is* complete *iff no completion rule is applicable to any assertion and if it contains no clash.*

2 Dependency Directed Backtracking

Non-deterministic completion rules are branching points for the expansion of an ABox. If a clash occurs in an ABox of inherently unsatisfiable concept terms, naive backtracking would cause thrashing by checking all possible branching points. *Dependency directed backtracking* (DDB) is an efficient technique which avoids thrashing because it only examines alternatives at branching points involved in a clash (see [4], [5]). For DDB each assertion has to be associated with a *dependency set* ($\boldsymbol{D}$) which contains *dependency numbers* denoting the branching points the assertion depends on.

Currently most of the optimized DL systems using DDB apply the trace technique, where each sub-tableau contains concepts asserted for the current individual and only these concepts are associated with dependency sets. The role and feature assertions do not have to be represented explicitly.[2] DLs with concrete domains have feature-complete sub-tableaux or no partitioning into sub-tableaux at all. Due to this a reasoner for a DL with concrete domains has to deal with sub-tableaux which contain several individuals in a sub-tableaux and with assertions instead of only concept terms.

[2] They are given by the recursive use of the satisfiability function.

define Compute_Dependency_Set
($New\_Assertion$, $Rule\_applied$, $Expanded\_Assertion$, $required\_Assertions$)

$\boldsymbol{D}_{New\_Assertion} := \boldsymbol{D}_{Expanded\_Assertion}$
for all $R\_Assertion$ in $required\_Assertions$ **do**
  $\boldsymbol{D}_{New\_Assertion} := \boldsymbol{D}_{New\_Assertion} \cup \boldsymbol{D}_{R\_Assertion}$
if non-deterministic-p ($Rule\_applied$) **then**
  increase($current\_Dependency\_Number$)
  $\boldsymbol{D}_{New\_Assertion} := \boldsymbol{D}_{New\_Assertion} \cup \{current\_Dependency\_Number\}$
return ($\boldsymbol{D}_{New\_Assertion}$)

Figure 1: Computation of dependency sets

2.1 Dependency Directed Backtracking for $\mathcal{ALC}(\mathcal{D})$

When applying DDB to $\mathcal{ALC}(\mathcal{D})$, the existing methods have to be adapted in two ways: (1) Concept assertions and also feature and predicate assertions have to be associated with dependency sets and (2) the capabilities of a tester for an admissible concrete domain have to be extended in order to allow DDB after *every* kind of clash. Calculi for DLs with concrete domains have (at least) four kinds of clashes after which backtracking might be necessary. A clash is always due to at least one culprit (Def. 2). Note that in contrast to other DLs, for $\mathcal{ALC}(\mathcal{D})$ (and therefore in all DLs with CDs) feature assertions and predicate assertions may be culprits as well. So for $\mathcal{ALC}(\mathcal{D})$ dependency sets assigned are not only concept assertions but also feature and predicate assertions.

Dependency numbers in the dependency set of an assertion identify branching points the assertion depends on. After a clash the union of dependency sets of all culprits $\boldsymbol{D}_{culprit}$ is built. The highest dependency number in $\boldsymbol{D}_{culprit}$ denotes the branching point to be backtracked to. If $\boldsymbol{D}_{culprit} = \emptyset$ the initial concept term is unsatisfiable. A procedure to compute the dependency set of a new assertion is shown in Figure 1. If a tableaux rule expands an assertion, say $a : (C \sqcap D)$, then the dependency set $\boldsymbol{D}_{a:(C \sqcap D)}$ is a subset of the dependency sets $\boldsymbol{D}_{a:C}$ and $\boldsymbol{D}_{a:D}$ of the new assertions. If the applied rule has required assertions, the dependency numbers of all required assertions are also dependency numbers for the new assertions. In case an assertion added by a rule with required assertions turns out to be a clash culprit, then the expanded assertion or a required assertion may be retracted from the ABox $\mathcal{A}$ in order to avoid the clash, if they depend on a branching point.

If a non-deterministic rule is applied during the tableaux expansion, a new branching point is encountered and a "fresh" dependency number is added to the dependency set of a new assertion.

If a fork $\{(a, b\_old) : f,\ (a, b\_new) : f\} \in \mathcal{A}$ is eliminated by replacing b_new with b_old in the ABox $\mathcal{A}$, the dependencies in $\boldsymbol{D}'_{(a,b\_old):f}$ must be computed:

If $(\boldsymbol{D}_{(a,b\_old):f} = \emptyset) \ \vee \ (\boldsymbol{D}_{(a,b\_new):f} = \emptyset)$
then $\boldsymbol{D'}_{(a,b\_old):f} = \emptyset$
else $\boldsymbol{D'}_{(a,b\_old):f} = \boldsymbol{D}_{(a,b\_old):f} \ \cup \ \{n \mid n \in \boldsymbol{D}_{(a,b\_new):f} \ \wedge \ n < max(\boldsymbol{D}_{(a,b\_old):f})\}$

If both dependency sets are empty, the feature assertion is not due to a choice at any branching point. In the other case the dependency numbers of $\boldsymbol{D}_{(a,b\_new):f}$ can be omitted, if they denote branching points encountered "later" than the branching point from which $(a, b\_old) : f$ originated.[3] Backtracking to one of these branching points would not retract the first feature assertion from the ABox.

Provers for DLs with admissible concrete domains consist of the DL reasoner and the CD consistency tester. In order to perform DDB after a concrete domain clash (Def. 2) it is necessary that the CD tester can identify all clash culprits. Such an *identifying concrete domain* tester, which finds all minimal, inconsistent sets of predicate assertions is the prerequisite for applying DDB efficiently to $\mathcal{ALC}(\mathcal{D})$ (and thereby for all DLs with a CD) after *every* kind of clash. It is not sufficient to identify just one set of inconsistent predicate assertions, as the Example 1 illustrates.

Example 1 *Assume adding the assertion $(x_1, x_2) : P_1$ with $\mathbf{D}_{(x_1,x_2):P_1} = \{1\}$ causes two concrete domain clashes in the ABox. The first clash results from the combination with $(x_1, y) : P_2$ and the second clash from the combination with $(x_2, z) : P_3$, where $\mathbf{D}_{(x_1,y):P_2} = \{5\}$ and $\mathbf{D}_{(x_2,z):P_3} = \{3\}$. If the CD tester finds only one minimal inconsistent set of predicate assertions, it may return the set $\{(x_1, x_2) : P_1, \ (x_2, z) : P_3\}$. The* DL *reasoner computes $\mathbf{D}_{Culprit} = \{1, 3\}$, backtracks to the branching point* 3 *and omits illegally the alternatives at branching point* 5.

However, an alternative to using an identifying CD tester is to restrict the application of DDB to the remaining three clash types. Then, in case of a concrete domain clash naive backtracking on the branching points, which added predicate assertions, must be performed. This assumes that a CD test is performed whenever a predicate assertion is added and would thereby lead to many more CD consistency tests and to thrashing in some cases.

For the most most admissible CDs of interest the complexity of a CD consistency test without identification of the culprits is at least in PSPACE and even worse, see [6]. So requiring an identifying CD and thus having an increase of complexity is a somewhat daunting result.

2.2 Dependency Directed Backtracking for $\mathcal{ALCRP}(\mathcal{D})$

The tableaux calculus for $\mathcal{ALCRP}(\mathcal{D})$ does not partition the ABox into subtableaux, so *all* kinds of assertions have to be associated with dependency sets.

[3] This branching point is denoted by the highest dependency number in $\boldsymbol{D}_{(a,b\_old):f}$.

In $\mathcal{ALCRP}(\mathcal{D})$ the RChoose rule is yet another non-deterministic rule, which introduces new dependency numbers. This rule adds alternatively a predicate assertion or its negated counterpart to the ABox. So it is to be expected that more CD tests will be necessary and many more concrete domain clashes will occur for $\mathcal{ALCRP}(\mathcal{D})$ than for $\mathcal{ALC}(\mathcal{D})$. So the use of an identifying concrete domain is even more mandatory for $\mathcal{ALCRP}(\mathcal{D})$. The dependency sets are determined and in case of fork elimination recomputed as described for $\mathcal{ALC}(\mathcal{D})$.

3 Model Merging and Caching

Model Merging (MM) is a very effective optimization technique for speeding up subsumption tests during the classification of a TBox. For each subsumption test ($C \sqsubseteq D$) a satisfiability test (sat?($C \sqcap \neg D$))) is performed. Each concept name mentioned in the TBox might be involved in many satisfiability tests, where the concept terms for C ($\neg C$) are expanded repetitively by the tableaux prover, if it is tested whether C is the subsumee (subsumer) of other concept names occurring in the TBox. MM re-uses the information (so-called *pseudo models*) cached from complete ABoxes, which were computed for previous subsumption tests by the tableaux prover. The MM technique is a sound but incomplete technique for testing the satisfiability of a conjunction of concepts by combining their pseudo models (see [4]). The conjunction $C \sqcap \neg D$ is satisfiable if the pseudo models of C and $\neg D$ can be merged by the MM algorithm. If these pseudo models cannot be merged, the tableaux prover must be applied for testing the satisfiability of the conjunction. MM algorithms for DLs with CDs based on the tableaux calculus, must take into account all clash types defined in Def 2 in order to determine if pseudo models are mergable.

We have developed algorithms for DLs with concrete domains for flat and deep MM as well. Due to the lack of space we only cover the flat model merging methods in this paper. For full details see [7].

3.1 Flat Model Merging

Flat model merging is used in the DL system FaCT for the description logic $\mathcal{ALC}f\mathcal{H}_{\mathcal{R}^+}$ (see [4]) and in RACE for the description logic $\mathcal{ALCNH}_{\mathcal{R}^+}$ (see [3]). Flat MM compares the set of atomic and negated atomic concepts asserted for the initial individual a and the assertions for the direct role and feature successors of a from the pseudo models to be merged. A *label set* $\mathcal{L}_{\mathcal{A}}(a)$ for an individual a is a set of all concept terms from all concept assertions for a in a completed ABox $\mathcal{A}$.

Def. 4 (Flat $\mathcal{ALC}(\mathcal{D})$ Pseudo Model) *Let D be an $\mathcal{ALC}(\mathcal{D})$ concept and let $\mathcal{A}$ be the ABox $\mathcal{A} = \{a : D\}$. The* flat $\mathcal{ALC}(\mathcal{D})$ pseudo model *M for D consists of the following* cache sets*:*

$\mathbf{S}^M_{\mathsf{CN}} = \{\mathsf{CN} | \mathsf{CN} \in \mathcal{L}_\mathcal{A}(a)\}, \quad \mathbf{S}^M_{\neg\mathsf{CN}} = \{\mathsf{CN} | \neg\mathsf{CN} \in \mathcal{L}_\mathcal{A}(a)\},$
$\mathbf{S}^M_{\forall R} = \{R \mid (\forall R.C) \in \mathcal{L}_\mathcal{A}(a)\}, \quad \mathbf{S}^M_{\exists R} = \{R \mid (\exists R.C) \in \mathcal{L}_\mathcal{A}(a)\},$
$\mathbf{S}^M_{\forall f} = \{f \mid (\forall f.C) \in \mathcal{L}_\mathcal{A}(a)\},$
$\mathbf{S}^M_{\exists f} = \{f \mid (\exists f.C) \in \mathcal{L}_\mathcal{A}(a)\} \cup \{f \mid f = first(u_i), u_i \ used \ in(\exists u_1, \ldots, u_n.P), (\exists u_1, \ldots, u_n.P) \in \mathcal{L}_\mathcal{A}(a)\}$

Flat pseudo models contain four cache sets with the names of roles or features "referring" to successors of the initial individual a of the ABox $\mathcal{A}$. There are two cache sets for roles (as well as for features) based on concept terms that trigger successor generating tableaux rules and that trigger non-generating rules. Flat MM is performed for two flat $\mathcal{ALC}(\mathcal{D})$ pseudo models M_1 and M_2 by testing the following conditions: Are there interactions for

1. the atomic concepts? $((\mathbf{S}^{M_1}_{\mathsf{CN}} \cap \mathbf{S}^{M_2}_{\neg\mathsf{CN}} \neq \emptyset) \vee (\mathbf{S}^{M_1}_{\neg\mathsf{CN}} \cap \mathbf{S}^{M_2}_{\mathsf{CN}} \neq \emptyset))$
2. role successors? $((\mathbf{S}^{M_1}_{\exists R} \cap \mathbf{S}^{M_2}_{\forall R} \neq \emptyset) \vee (\mathbf{S}^{M_1}_{\forall R} \cap \mathbf{S}^{M_2}_{\exists R} \neq \emptyset))$
3. feature successors?
$((\mathbf{S}^{M_1}_{\exists f} \cap \mathbf{S}^{M_2}_{\exists f} \neq \emptyset) \vee (\mathbf{S}^{M_1}_{\exists f} \cap \mathbf{S}^{M_2}_{\forall f} \neq \emptyset) \vee (\mathbf{S}^{M_1}_{\forall f} \cap \mathbf{S}^{M_2}_{\exists f} \neq \emptyset))$

If all three conditions are false there is no interaction between the pseudo models, they can be merged and the conjunction of concept terms is satisfiable. If one of the conditions is true, the tableaux prover must be used to test the satisfiability of the conjunction.

3.1.1 Flat Model Merging for $\mathcal{ALCRP}(\mathcal{D})$

The applicability of MM for $\mathcal{ALCRP}(\mathcal{D})$ is limited due to the phantom edges caused by value restrictions for complex roles. The problem is that a new phantom edge may be established "across" the complete ABoxes the pseudo models are derived from, as the following example illustrates.

Example 2 *Assume we test if C_1 and C_2 are satisfiable as a conjunction, with* $C_1 \doteq \exists R_1.\ (\ (\exists f_1 \circ f_2, f_3.P_2) \sqcap\ (\forall\ (\exists (f_1 \circ f_2, f_3)(f_4, f_3 \circ f_1).P_1)\ .\neg D))$ $C_2 \doteq \exists R_2.\ (D\ \sqcap\ (\exists f_4, f_3 \circ f_1.P_3))$. *We derive ABoxes for C_1, C_2 and $C_1 \sqcap C_2$ as shown in Figure 2. However, the ABox of $C_1 \sqcap C_2$ contains a phantom edge causing a primitive clash at the R_2 successor individual. Neither the phantom edge nor the clash can be detected by flat MM.*

MM is therefore only applicable to $\forall R_c$-free models.[4] A model is $\forall R_c$*-free* iff it is cached from a complete ABox $\mathcal{A}$ where no individual a and no complex role R_c exists such that $a : (\forall R_c\,.\,C) \in \mathcal{A}$. Please note that restricting MM to $\forall R_c$-free models does not restrict it to concept terms without value restriction for

[4]Even deep model merging, the recursive variant of flat MM, cannot handle the phantom edges in a efficient way.

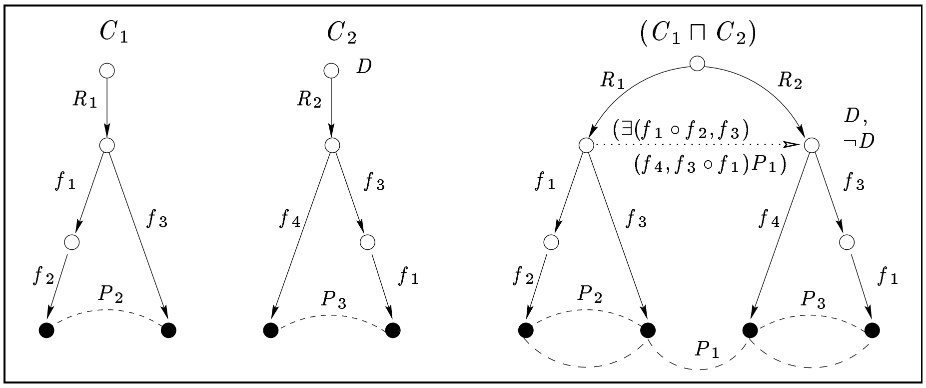

Figure 2: Phantom edge across models

complex roles. If a concept term uses such a value restriction only in disjunctions, MM might still be used for the resulting model.

In contrast to value restrictions, exists restrictions for complex roles can be processed by flat MM. In a first step the tableaux rules expanding such a term add a new role successor individual for the complex role to the ABox. In a second step the chains of the role predecessor (the u_i's), the chains of the role successor (the v_i's) and the predicate assertion are added. Because the role successor is a new individual in the ABox all of its feature successors are also new. Therefore, not both $\forall R_c$-free pseudo models to be merged can refer to the same role successor individual for a complex role. Interactions can only occur at the predecessor individual and along its feature chains. Therefore the role predecessor individual and its feature-chains must be checked by MM. Flat pseudo models for $\mathcal{ALCRP}(\mathcal{D})$ need to cache the first features of the u_i's appearing in terms like $(\forall\,(\exists(u_1,\ldots,u_n)(v_1,\ldots,v_m).P)\,.\,C)$ in the cache set $\mathbf{S}^{M}_{\exists f}$. If these first features are added to the flat pseudo model, the same three test conditions as for $\mathcal{ALC}(\mathcal{D})$ can be applied to perform flat MM for $\mathcal{ALCRP}(\mathcal{D})$.

4 Conclusion and Future Work

Experiences with other DLs have shown that the classification of realistic application KBs demands provers supporting at least MM and DDB. Practical applications (e.g. configuration tasks) ask for reasoning with CD objects such as real numbers. We have shown that two of the most effective optimization techniques can be adapted to the satisfiability test for $\mathcal{ALC}(\mathcal{D})$, which is the basic logic of DLs with concrete domains. Unfortunately these results do not hold for $\mathcal{ALCRP}(\mathcal{D})$ concepts in general. Only concepts with $\forall R_c$-free models can be processed by MM.

Applying DDB to concrete domain clashes requires identifying CD testers. Unless an identifying CD is used, the CD consistency test must be performed after each tableaux rule which generates a predicate assertion. Without an identifying concrete domain DDB can only be performed after the remaining three clash types. Using an identifying CD the "expensive" CD consistency test does not have to be performed after every tableaux rule adding a predicate assertion. This way of reducing the numbers of CD tests might lessen the increase of computational effort due to the identification of all inconsistent sets of predicate assertions. Which of these two ways of DDB turns out to be more efficient, depends on the CD used and requires an implementation and empirical testing of these methods by means of KBs using DLs with CDs.

A next step to further optimize provers for DLs with CDs is to investigate dynamic backtracking. This backtracking technique does not discard intermediate results when backtracking to a branching point. So for DLs with CDs (and especially $\mathcal{ALCRP}(\mathcal{D})$) this method avoids the re-computing of satisfiable sets of predicate assertions.

Acknowledgment

We would like to thank Ralf Möller and Michael Wessel for helpful discussions and suggestions. We also thank the anonymous reviewers for valuable comments on this paper.

References

[1] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Twelfth International Conference on Artificial Intelligence, Darling Harbour, Sydney, Australia*, pages 452–457, August 1991.

[2] V. Haarslev, C. Lutz, and R. Möller. A description logic with concrete domains and a role-forming predicate operator. *Journal of Logic and Computation*, 9:351–384, June 1999.

[3] V. Haarslev and R. Möller. Optimizing TBox and ABox reasoning with pseudo models, August 2000. In this volume.

[4] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics.* PhD thesis, University of Manchester, 1997.

[5] I. Horrocks and P. Patel-Schneider. Optimising description logic subsumption. *Journal of Logic and Computation*, 9:267–293, June 1999.

[6] Carsten Lutz. The complexity of reasoning with concrete domains (revised version). LTCS-Report 99-01, LuFg Theoretical Computer Science, RWTH Aachen, Germany, 1999.

[7] A.-Y. Turhan. Optimization methods for the satisfiability test for description logics with concrete domains. Master's thesis, University of Hamburg, Computer Science Department, April 2000.