Concept Projection in Algebras for Computing Certain Answer Descriptions

Jeffrey Pound, David Toman, Grant Weddell and Jiewen Wu

Cheriton School of Computer Science, University of Waterloo, Canada
{jpound, david, j55wu, gweddell}@uwaterloo.ca

1 Introduction

We introduce a projection operator over concepts in a given description logic that produces least subsumers of concepts in a language fragment that satisfies additional syntactic requirements specified as a parameter of the operator. The operator complements existing operators for concept selection and concept ordering that have been proposed in earlier work [3, 4]. We show how, altogether, the operators can form the basis of an algebra for computing more informative and user friendly varieties of certain answers over what we call a *description base*. This is a *description logic* (DL) knowledge base in which the ABox is replaced by a fixed and finite collection of arbitrary concepts.

In the sections that follow, we introduce the notion of a description base, of a *certain answer description* over a description base and then present an algebra for computing a simple variety of certain answer descriptions that incorporates the new projection operator. We elaborate on the semantics of this operator, in particular the notion of a *projection description*, in the remaining sections of the paper where we consider the operator's semantics as well as methods for efficient evaluation. In the final section, we outline a motivating example and suggest useful ways in which both projection descriptions and our query algebra might be extended.

1.1 Certain Answer Descriptions and Description Bases

Assume $\mathcal{K}$ $(= (\mathcal{T}, \mathcal{A}))$ denotes a knowledge base in which concepts conform to a given DL, and consider the purpose served by $\mathcal{K}$'s ABox $\mathcal{A}$ when it comes to computing certain answers over $\mathcal{K}$ to a conjunctive query of the form

$$Q(x_1, ..., x_k) \leftarrow \mathit{QueryBody},$$

where *QueryBody* in turn has the form

$$\exists x_{k+1}, ..., \exists x_m : \Big(\bigwedge C(x_i)\Big) \wedge \Big(\bigwedge R(x_i, x_j)\Big)$$

in which the predicate names correspond to concepts and roles in the DL. Most importantly, the ABox supplies a fixed and finite collection of constant symbols $\{a_1, ..., a_n\}$ which in turn defines a space of n^k *possible answers* to Q: k-tuples θ_i of the form "$\{(x_1, a_{i_1}), ..., (x_k, a_{i_k})\}$" in which query variables x_j are paired with constant symbols a_{i_j}. Viewed as a substitution, recall that θ_i is a *certain answer* to Q exactly when $\mathcal{K} \models \mathit{QueryBody}\ \theta_i$. Remember, however, that constant symbols and certain answers are still syntactic artifacts and that the former are not actual domain elements in an interpretation.

Now assume the DL supports *nominals*, $\{a\}$. Answers θ_i can then have the alternative form

$$\{(x_1, \{a_{i_1}\}), ..., (x_k, \{a_{i_k}\})\} \tag{1}$$

in which query variables x_j are now paired with nominal concepts $\{a_{i_j}\}$. In this case, (1) will be a certain answer exactly when

$$\mathcal{K} \models \forall x_1, ..., \forall x_k : (\{a_{i_1}\}(x_1) \wedge ... \wedge \{a_{i_k}\}(x_k) \rightarrow \mathit{QueryBody}). \tag{2}$$

Among other things, nominals allow one to add inclusion dependencies $\{a_i\} \sqsubseteq C$ and $\{a_i\} \sqsubseteq \exists R.\{a_j\}$ to $\mathcal{T}$ that correspond to ABox assertions $C(a_i)$ and $R(a_i, a_j)$ in $\mathcal{A}$, respectively. An ABox can then be replaced by a fixed and finite collection of nominals $\{\{a_1\}, ..., \{a_n\}\}$ since its only remaining purpose is to list the nominal concepts that can occur in (1) above, and therefore in the antecedent of the implication in (2) above.

From a user's perspective, this characterization of answers generalizes in an appealing way when one allows the collection of nominals that replaces an ABox to consist instead of a fixed and finite collection of *arbitrary* concepts $\{C_1, ..., C_n\}$. In this way, the concepts appearing in (1) can provide more informative descriptions to the user that *necessarily* describe x_i-components of answer k-tuples. In such circumstances, and for the remainder of the paper, we refer to this list of arbitrary concepts as a DBox or *description box*, to a knowledge base $\mathcal{K}$ that replaces an ABox with a DBox as a *description base*, and to certain answers that pair query variables with arbitrary concepts as *certain answer descriptions*.

Note that, for a description base $\mathcal{K}$ to qualify as consistent, it is sensible to add the condition that there exists a model for which the interpretation of each concept in $\mathcal{K}$'s DBox is nonempty in addition to the standard requirement that the model satisfies each inclusion dependency in $\mathcal{K}$'s TBox. This ensures that concepts describing an x_i-component of an answer k-tuple are never vacuous, indeed, that each concept in a DBox is there because it describes *some* entity of interest to a user. Such a condition can always be captured in a TBox by adding inclusion dependencies of the form "$\top \sqsubseteq \exists R.C_i$", where R is a fresh role name.

1.2 Towards an Algebra for Description Bases

Returning to the standard notion of $\mathcal{K}$ as a knowledge base, consider the subclass of queries having the form "$Q(x) \leftarrow C(x)$". In this simplest case, answers may elide any mention of the single query variable x, and correspond more simply to the constant symbols a_i that must satisfy a *QueryBody* with conditions that can be *rolled up* in a single concept C. For a description base, when $\mathcal{K} = (\mathcal{T}, \{C_1, ..., C_n\})$, such queries return *unary* certain answer descriptions, that is, each concept C_i for which $\mathcal{T} \models C_i \sqsubseteq C$.

The basis of an algebra for computing unary certain answer descriptions has been proposed in earlier work [3, 4]. The algebra consists of three operators given by the following grammar for a query Q.

$$Q \quad ::= \quad \mathcal{K} :: Od \mid \sigma_C(Q) \mid \textsc{Sort}_{Od}(Q)$$

Since the focus of this earlier work was on performance and scalability, the first priority was to introduce operators that could express such things as an *index scan* or a *sort*, operations that are fundamental to issues of efficiency in database query evaluation. In particular, this entailed the development of a theory of *ordering descriptions* that could be used to characterize strict partial orders over the concepts generated by a given DL (the "*Od*" part of the first and third operators are ordering descriptions) together with the notion of a binary search tree index of concepts called a *description index.* Altogether, this enabled a formal consideration of sorting and comparison based search over a description base DBox.

The first operator has two purposes: it defines a description index of the concepts that occur in the DBox of a given description base, and it computes a list of concepts that corresponds to an inorder traversal of the index. The second operator filters concepts occurring in a concept list by retaining only those subsumed by a given parameter concept. The third operator sorts a concept list according to a partial order defined by a given ordering description.

An outstanding issue with this algebra is that it is necessary to supply, a priori, all concepts that might be of interest to any user in the DBox of a description base. Since users will want concepts that reflect different external views and interests, the DBox will almost certainly require multiple concepts for the same underlying entity e together with a means to distinguish among the possible concepts for e. In this paper, we propose to address this problem by adding a new *projection* operator to the grammar for the above algebra.

$$Q \quad ::= \quad \mathcal{K} :: Od \mid \sigma_C(Q) \mid \textsc{Sort}_{Od}(Q) \mid \pi_{Pd}(Q) \tag{3}$$

The operator can be used to rewrite concepts produced by a subquery Q to subsuming concepts that satisfy an additional syntactic requirement given by the Pd part of the new operator. We call the syntactic requirement a *projection description.* The new operator replaces each concept in the concept list produced by its subquery by a least subsuming concept that satisfies the projection description.

As in earlier work on ordering descriptions, our main objective is to develop a theory of projection descriptions that will work for a variety of DLs that satisfy basic requirements. The first is that any qualifying DL has $\mathcal{EL}$ as a subdialect. One important reason is that $\mathcal{EL}$ concepts are a close approximation to nested relations and to XML and are therefore an ideal starting point for user friendly descriptions. Another reason is that we can show that π_{Pd} computes *least subsumers* of argument concepts with respect to an $\mathcal{EL}$ fragment satisfying Pd. We also assume that any qualifying DL will have at least one concrete domain that supports constant terms and an underlying total order. These constant terms can then be used in descriptions that correspond to the notion of a tuple in the relational model. As a consequence, our projection operator then has relational tuple projection as a special case. In addition, the operator naturally accommodates nested relations and their projections as well.

We formally define projection descriptions in the next section, and then proceed to consider how projection operations can be efficiently implemented by appealing only to concept subsumption in a given DL, e.g., by using classifications of concepts that occur in a given terminology.

2 Projection Descriptions

To introduce projection descriptions, we use the DL $\mathcal{ALC}(\mathbf{D})$ in which the concrete domain $\mathbf{D}$ supports equality and comparison operations over an underlying domain consisting of rationals and strings.[1] However, to reiterate, any DL that contains $\mathcal{EL}(\mathbf{D})$ would qualify.

Definition 1 (Description Logic $\mathcal{ALC}(\mathbf{D})$). *Let* $\{A, A_1, \ldots\}$*,* $\{R, R_1, \ldots\}$*,* $\{f, g, f_1, \ldots\}$*, and* $\{k, k_1, \ldots\}$ *denote sets of* primitive concepts*,* roles*,* concrete features*, and* constants *respectively. A* concept *is defined by the grammar:*

$$\begin{array}{rl} C, D ::= & \top \mid \bot \mid A \mid \neg C \mid C \sqcap D \mid \exists R.C \\ \mid & f = k \qquad (\textit{equality over } \triangle_{\mathbf{D}}) \\ \mid & f < g \qquad (\textit{linear order over } \triangle_{\mathbf{D}}) \end{array}$$

An inclusion dependency *has the form $C \sqsubseteq D$. A* terminology *or* TBox *$\mathcal{T}$ is a finite set of inclusion dependencies.*

An interpretation $\mathcal{I}$ is a 2-tuple $(\triangle, \cdot^{\mathcal{I}})$ in which $\triangle$ is a domain that contains $\triangle_{\mathbf{D}}$, the set of rationals and finite strings, as a proper subset. We write $\triangle_{\mathbf{A}}$ to denote the abstract *domain $\triangle \setminus \triangle_{\mathbf{D}}$. Also, $\cdot^{\mathcal{I}}$ is an interpretation function that maps each concrete feature f to a total function $(f)^{\mathcal{I}} : \triangle_{\mathbf{A}} \to \triangle_{\mathbf{D}}$, each role R to a relation $(R)^{\mathcal{I}} \subseteq (\triangle_{\mathbf{A}} \times \triangle_{\mathbf{A}})$, each primitive concept A to a set $(A)^{\mathcal{I}} \subseteq \triangle_{\mathbf{A}}$, the "$=$" symbol to the equality relation on $\triangle_{\mathbf{D}}$, the "$<$" symbol to the binary relation for the linear order on $\triangle_{\mathbf{D}}$, and each constant k to a constant in $\triangle_{\mathbf{D}}$. The interpretation function is extended to arbitrary concepts in the standard way.*

An interpretation $\mathcal{I}$ satisfies an inclusion dependency $C \sqsubseteq D$ if $(C)^{\mathcal{I}} \subseteq (D)^{\mathcal{I}}$, and $\mathcal{T} \models C \sqsubseteq D$ if $(C)^{\mathcal{I}} \subseteq (D)^{\mathcal{I}}$ for any interpretation $\mathcal{I}$ that satisfies all inclusion dependencies in $\mathcal{T}$.

We use standard abbreviations such as $C \sqcup D$ for $\neg(\neg C \sqcap \neg D)$ and $f \leq k$ for $(f = k) \sqcup ((f < g) \sqcap (g = k))$. Also, given a finite set S of $\mathcal{ALC}(\mathbf{D})$ concepts, we write $\sqcap S$ to denote $\top$ if S is empty and the concept $D_1 \sqcap \cdots \sqcap D_n$ otherwise, when $S = \{D_1, ..., D_n\}$.

With the presumption of $\mathcal{ALC}(\mathbf{D})$, the syntax and semantics of our new projection operation are given by the following.

[1] See [1] for an excellent introduction to description logics.

Definition 2 (Projection Description). *Let L be a DL that includes $\mathcal{ALC}(\mathbf{D})$, possibly without negation, and f, R and C any concrete feature, role and concept in L, respectively. A* projection description *Pd is defined by the grammar:*

$$Pd \ ::= \ C? \ \mid \ C^{\uparrow} \ \mid \ f \ \mid \ Pd_1 \sqcap Pd_2 \ \mid \ Pd_1 \cup Pd_2 \ \mid \ \exists R.Pd \qquad (4)$$

Now let $\alpha(Pd)$ be defined as follows:

$$\alpha(Pd) \ = \ \begin{cases} \emptyset & \text{if } Pd = \text{“}C?\text{”}, \text{“}C^{\uparrow}\text{” or “}f\text{”}; \\ \{R\} & \text{if } Pd = \text{“}\exists R.Pd_1\text{”; and} \\ \alpha(Pd_1) \cup \alpha(Pd_2) & \text{if } Pd = \text{“}Pd_1 \sqcap Pd_2\text{” or “}Pd_1 \cup Pd_2\text{”}. \end{cases}$$

Then Pd is well formed *if, for any subexpression $Pd_1 \sqcap Pd_2$ or $Pd_1 \cup Pd_2$, $\alpha(Pd_1) \cap \alpha(Pd_2) = \emptyset$.*

The condition that Pd is well-formed ensures that requests for information concerning a given role are never *distributed* in different parts of an answer (sub) tuple. This is a necessary condition to help combat combinatorial effects that would otherwise make projection involving roles infeasible.

Definition 3 (Induced Concepts). *Let L be a DL that includes $\mathcal{ALC}(\mathbf{D})$, possibly without negation, and Pd a projection description over roles in L. We define the sets $L_{|Pd}$ and $L^{\text{TUP}}_{|Pd}$, the L* concepts generated by *Pd and L* tuple concepts generated by *Pd, respectively, as follows:*

$$L_{|Pd} \ = \ \{\sqcap S \mid S \subseteq_{\text{fin}} L^{\text{TUP}}_{|Pd}\}, \text{ and}$$

$$L^{\text{TUP}}_{|Pd} \ = \ \begin{cases} \{C\} & \text{if } Pd = \text{“}C?\text{”}; \\ \mathbf{A} & \text{if } Pd = \text{“}C^{\uparrow}\text{”}; \\ \{f = k \mid k \in \triangle_{\mathbf{D}}\} & \text{if } Pd = \text{“}f\text{”}; \\ \{C_1 \sqcap C_2 \mid C_1 \in L^{\text{TUP}}_{|Pd_1} \wedge C_2 \in L^{\text{TUP}}_{|Pd_2}\} & \text{if } Pd = \text{“}Pd_1 \sqcap Pd_2\text{”}; \\ L^{\text{TUP}}_{|Pd_1} \cup L^{\text{TUP}}_{|Pd_2} & \text{if } Pd = \text{“}Pd_1 \cup Pd_2\text{”; and} \\ \{\sqcap S \mid S \subseteq_{\text{fin}} \{\exists R.C \mid C \in L_{|Pd_1}\}\} & \text{if } Pd = \text{“}\exists R.Pd_1\text{”}, \end{cases}$$

where $\mathbf{A}$ is the set of all primitive concepts in the alphabet of L.

Note that, while in general the language $L_{|Pd}$ is infinite, for every fixed and finite terminology $\mathcal{T}$ and concept C, the language $L_{|Pd}$ restricted to the symbols used in $\mathcal{T}$ and C is necessarily finite.

Definition 4 (Projection Semantics). *The semantics of the new projection operator is given as follows: query $\pi_{Pd}(Q)$ replaces each concept C in the concept list computed by Q by a* least subsumer *of C in $L_{|Pd}$.*

The remainder of this section begins to consider how least subsumers in $L_{|Pd}$ can be effectively computed when Pd is well-formed. To start, we introduce the notion of a *role path* that helps to simplify manipulating concepts of the form $\exists R_1. \ldots .\exists R_n.C$.

Definition 5 (Role Path). *Let L be a DL that includes $\mathcal{ALC}(\mathbf{D})$, possibly without negation, and R be any role in L. A* role path *Rp is defined by the grammar:*

$$Rp \ \ ::= \ \ Id \ \mid \ Rp.R$$

The expression $\exists Rp.C$ denotes the concept C when $Rp =$ "Id", and the concept $\exists Rp_1.\exists R.C$ otherwise, when $Rp =$ "$Rp_1.R$".

Definition 6 (Projection Function). *Let L be a DL that includes $\mathcal{ALC}(\mathbf{D})$, possibly without negation, and $\mathcal{T}$, C, Rp and Pd a respective TBox, concept, role path and projection description in L. The* functional projection of C for Pd with respect to $\mathcal{T}$ and Rp*, written $(Pd)_{\mathcal{T},Rp}(C)$, is a finite set S of L concepts satisfying the following conditions.*

Case $Pd =$ "$C_1?$": *$S = \{C_1\}$ if $\mathcal{T} \models C \sqsubseteq \exists Rp.C_1$, and $\emptyset$ otherwise.*
Case $Pd =$ "$C_1^{\uparrow}$":

$$S = (A_1? \cup \cdots \cup A_n?)_{\mathcal{T},Rp}(C),$$

where $\{A_1, ..., A_n\}$ are all primitive concepts A occurring in $\mathcal{T}$ and C such that $\mathcal{T} \not\models \exists Rp.A \sqsubseteq \exists Rp.C_1$; $S = \emptyset$ for $n = 0$.

Case $Pd =$ "f":

$$S = ((f = k_1)? \cup \cdots \cup (f = k_n)?)_{\mathcal{T},Rp}(C),$$

where $\{k_1, ..., k_n\}$ are all constants occurring in $\mathcal{T}$ and C; $S = \emptyset$ for $n = 0$.

Case $Pd =$ "$Pd_1 \sqcap Pd_2$":

$$S = \{D_1 \sqcap D_2 \mid (\forall i \in \{1,2\} : D_i \in (Pd_i)_{\mathcal{T},Rp}(C)) \wedge \mathcal{T} \models C \sqsubseteq \exists Rp.(D_1 \sqcap D_2)\}.$$

Case $Pd =$ "$Pd_1 \cup Pd_2$": $S = (Pd_1)_{\mathcal{T},Rp}(C) \cup (Pd_2)_{\mathcal{T},Rp}(C)$.
Case $Pd =$ "$\exists R.Pd_1$":

$$S = \lfloor \{\exists R.(\sqcap S') \mid S' \subseteq (Pd_1)_{\mathcal{T},Rp.R}(C) \wedge \mathcal{T} \models C \sqsubseteq \exists Rp.\exists R.(\sqcap S')\} \rfloor_{\mathcal{T},Rp}.$$

In the final case, we write $\lfloor S \rfloor_{\mathcal{T},Rp}$ to denote the reduction of S with respect to $\mathcal{T}$ and Rp*, that is, the set of all $D_1 \in S$ such that there is no $D_2 \in S$ for which $\mathcal{T} \models \exists Rp.D_2 \sqsubseteq \exists Rp.D_1$ and $\mathcal{T} \not\models \exists Rp.D_1 \sqsubseteq \exists Rp.D_2$.*

Our main result now follows, in which functional projection is related to least subsumers.

Theorem 1. *Let L be a DL that includes $\mathcal{ALC}(\mathbf{D})$, possibly without negation, and $\mathcal{T}$, C and Pd a respective terminology, concept and well-formed projection description in L. Then $\sqcap \lfloor (Pd)_{\mathcal{T},Id}(C) \rfloor_{\mathcal{T},Id}$ is a least subsumer of C in $L_{|Pd}$.*

Proof (outline). By induction on the structure of Pd.

Note that the restriction to $L_{|Pd}$ is essential to guarantee that the least subsumer always exists (otherwise, least subsumers may not exist [2]).

3 Computing Projection Descriptions

A top-level procedure for computing $(Pd)_{\mathcal{T},Rp}(C)$ is given by a definition of Project in Figure 1. Clearly, a straightforward coding for the procedures invoked by Project to handle each of the six conditions follows directly from Definition 6. However, it is evident that this simple coding strategy for many of

```
PROJECT($Pd, \mathcal{T}, Rp, C$)
switch on $Pd$
    case "$C_1$?": return PROJECTCONCEPT($C_1, \mathcal{T}, Rp, C$)
    case "$C_1^{\uparrow}$": return PROJECTATOMICCONCEPTS($C_1, \mathcal{T}, Rp, C$)
    case "$f$": return PROJECTFEATURE($f, \mathcal{T}, Rp, C$)
    case "$Pd_1 \sqcap Pd_2$": return PROJECTJOIN($Pd_1, Pd_2, \mathcal{T}, Rp, C$)
    case "$Pd_1 \cup Pd_2$": return PROJECT($Pd_1, \mathcal{T}, Rp, C$) $\cup$ PROJECT($Pd_2, \mathcal{T}, Rp, C$)
    case "$\exists R.Pd_1$": return PROJECTROLE($Pd_1, R, \mathcal{T}, Rp, C$)
```

Fig. 1. COMPUTING PROJECTIONS AT THE TOP-LEVEL

these procedures will lead to an unacceptably large number of subsumption tests. In particular, the resulting PROJECTATOMICCONCEPTS procedure for the "$C^{\uparrow}$" case will require a number of such tests proportional to the number of primitive concepts in the terminology, or much worse: exponential in the number of primitive concepts, e.g., when called indirectly by a naïvely coded PROJECTROLE procedure. This happens, for example, with a projection description of the form "$\exists R.(\bot^{\uparrow})$".

A naïvely coded PROJECTJOIN procedure for the "$Pd_1 \sqcap Pd_2$" case is also problematic since it requires a number of subsumption tests equal to the product of the number of descriptions computed by Pd_1 and Pd_2. This circumstance quickly becomes intolerable for iterated uses of this procedure such as for projection descriptions of the form "$(f_1 \sqcap \cdots \sqcap f_n)$" that compute descriptions of relational tuples, cases that are likely to occur in practice.

While these performance issues are unavoidable in the worst case, it is possible to improve the performance of projection computation for many situations. Starting with procedure PROJECTJOIN, we now outline algorithms for the procedures invoked by PROJECT in Figure 1 that will have much better performance in situations that are more likely to occur in practice.

Procedure PROJECTJOIN

Our algorithm for this procedure follows a "nested loops" strategy in which concepts computed by an outer loop may be used to further qualify concepts computed by an inner loop. A simple way to accomplish this is to replace Definition 5 above with a more general notion of a role path that enables sideways communication of outer loop concepts.

Definition 7 (General Role Path). *Let L be a DL that includes $\mathcal{ALC}(\mathbf{D})$, possibly without negation, and R and C be any role or concept in L, respectively. A* general role path *Rp is defined by the grammar:*

$$Rp \quad ::= \quad Id \ \mid \ Rp.R \ \mid \ Rp.C$$

The expression $\exists Rp.C$ denotes the concept C when $Rp =$ "Id", the concept $\exists Rp_1.\exists R.C$ when $Rp =$ "$Rp_1.R$", and the concept $\exists Rp_1.(C_1 \sqcap C)$ otherwise, when $Rp =$ "$Rp_1.C_1$".

Algorithm 1: $\textsc{ProjectJoin}(Pd_1, Pd_2, \mathcal{T}, Rp, C)$

```
$result \leftarrow \emptyset$
foreach $C_1 \in \textsc{Project}(Pd_1, \mathcal{T}, Rp, C)$ do
    foreach $C_2 \in \textsc{Project}(Pd_2, \mathcal{T}, Rp.C_1, C)$ do
        $result \leftarrow result \cup \{(C_1 \sqcap C_2)\}$
return $result$
```

There are additional opportunities for improving the performance of Algorithm 1. For one, the procedure might explore alternative permutations of projection descriptions with the form "$(Pd_1 \sqcap \cdots \sqcap Pd_n)$" that might result in a more efficient nesting order. For another, the procedure may opt to only append C_1 to Rp in the inner loop if the overall cost of subsumption checks is expected to decrease.

Procedure $\textsc{ProjectAtomicConcepts}$

The expected time for this procedure can be improved considerably by employing a classification of the primitive concepts in a given terminology. In particular, one conducts a depth-first-search of the classification, taking advantage of pruning opportunities when parent atomic concepts in the classification are disqualified. An implementation of this procedure given by Algorithm 2 below assumes the following definition.

Definition 8 (TBox Classification). *Let $\mathcal{T}$ be a TBox in $\mathcal{ALC}(\mathbf{D})$. A* classification *of $\mathcal{T}$ is a directed acyclic graph G $(= (N, E))$ with nodes N corresponding to the primitive concepts in $\mathcal{T}$ and with E consisting of all edges (A_1, A_2) such that:*

- $\mathcal{T} \models A_1 \sqsubseteq A_2$,
- $\mathcal{T} \not\models A_2 \sqsubseteq A_1$, *and*
- *there is no distinct $A_3 \in N$ such that $\mathcal{T} \models A_1 \sqsubseteq A_3$ and $\mathcal{T} \models A_3 \sqsubseteq A_2$.*

We write N_G (resp. E_G) to denote the nodes (resp. edges) of G, where the terminology is assumed by context.

Again, there are opportunities for improving the performance of Algorithm 2. For example, it would help to ensure the performance gains obtained by pruning during depth-first-search if one adds new internal primitive concepts to the classification if there are a large number of root nodes, or to split cases in which a large number of edges would otherwise lead to an existing node.

Procedure $\textsc{ProjectRole}$

The naïve implementation of this procedure involves quantifying over all subsets of concepts computed by the evaluation of the nested projection description. However, for a given terminology $\mathcal{T}$, role path Rp and concept C, consider the following:

Algorithm 2: ProjectAtomicConcepts($C_1, \mathcal{T}, Rp, C$)

```
$stack \leftarrow \emptyset$
$result \leftarrow \emptyset$
// Initialize stack with the roots of the classification of $\mathcal{T}$
foreach node $A \in N_G$ with no outgoing edges do
    $stack$.Push($A$)
// Perform a depth first search of the classification
while $stack$.NotEmpty() do
    $A \leftarrow stack$.Pop()
    if $\mathcal{T} \models C \sqsubseteq \exists Rp.A$ and $\mathcal{T} \not\models \exists Rp.A \sqsubseteq \exists Rp.C_1$ then
        $result \leftarrow result \cup \{A\}$
        foreach $(A', A) \in E_G$ do
            $stack$.Push($A'$)
foreach $A$ occurring in $C$ such that $A \notin N_G$ and $\mathcal{T} \models C \sqsubseteq \exists Rp.A$ do
    $result \leftarrow result \cup \{A\}$
return $result$
```

1. It is less likely that $\mathcal{T} \models C \sqsubseteq \exists Rp.(\sqcap S)$, for concept sets S with larger numbers of elements; and
2. For any pair of concept sets S_1 and S_2, if $\mathcal{T} \not\models C \sqsubseteq \exists Rp.(\sqcap S_1)$, then $\mathcal{T} \not\models C \sqsubseteq \exists Rp.(\sqcap (S_1 \cup S_2))$.

These observations motivate the implementation of ProjectRole given by Algorithm 3. The algorithm uses a queue to conduct a breadth-first-search of subsets in such a way that ensures a "frontier" of disqualified sets will prune containing candidate sets. Note that, to avoid considering more than one permutation of a candidate, the procedure assumes a total lexicographic order "$\prec$" over all concepts.

The algorithm works particularly well in cases where the argument projection description parameter Pd has the form "$f \sqcap Pd$" such as in the case of projection descriptions that compute descriptions of sets of relational tuples. In this case, the number of subsumption checks performed directly by the algorithm is quadratic in the size of result computed by the recursive call to Project in the first line.

Procedures ProjectConcept**,** ProjectFeature **and** Reduce
An implementation of the remaining procedures is straightforward. In the case of ProjectConcept, it clearly suffices to mirror the corresponding case specification in Definition 2. An efficient algorithm for ProjectFeature requires only a bit more thought: one proceeds by a progressive refinement of intervals over the concrete domain, narrowing by binary search on intervals to the required set of concepts of the form "$f = k$". Finally, an implementation of procedure Reduce is given by Algorithm 4.

Algorithm 3: $\textsc{ProjectRole}(Pd, R, \mathcal{T}, Rp, C)$

```
$S_1 \leftarrow \textsc{Project}(Pd, \mathcal{T}, Rp.R, C)$
// Check empty set cases
if $S_1 = \emptyset$ then
    if $\mathcal{T} \models C \sqsubseteq \exists(Rp.R).(\top)$ then
        return $\{\exists R.(\top)\}$
    return $\emptyset$
$result \leftarrow \emptyset$
$queue \leftarrow \emptyset$
foreach $C_1 \in S_1$ do
    $queue.\textsc{Enqueue}(\{C_1\})$
while $queue.\textsc{NotEmpty}()$ do
    $S_2 \leftarrow queue.\textsc{Dequeue}()$
    $notgrown \leftarrow true$
    foreach $C_1 \in S_1$ where for all $C_2 \in S_2 : C_2 \prec C_1$ do
        if $\mathcal{T} \models C \sqsubseteq \exists(Rp.R)(C_1 \sqcap (\sqcap S_2))$ then
            $queue.\textsc{Enqueue}(\{C_1\} \cup S_2)$
            $notgrown \leftarrow false$
    if $notgrown$ then
        $result \leftarrow result \cup \{\exists R.(\sqcap S_2)\}$
return $\textsc{Reduce}(result, \mathcal{T}, Rp)$
```

4 Discussion

An application that serves as a guiding influence on this work relates to a hypothetical description base system to which an internet browser submits queries in our algebra. The queries originate in the web pages for an enterprise, and are replaced on the fly by the browser with the resulting lists of $\mathcal{EL}$-like concepts (suitably mapped to HTML) that are returned by the system in response. For example, consider the case of an online supplier of photography equipment. As part of a web presence, the supplier maintains a description base $\mathcal{K}$ of concepts that describe items that are available for purchase as well as web pages with embedded queries over this description base. One query Q in some web page might have the form "$\pi_{Pd}(\textsc{Sort}_{Price}(\sigma_C(\mathcal{K} :: Price)))$" in which the selection condition C is the concept "$ProductCode =$ "digicam" $\sqcap\, Price < 300$" and where the projection description is given by

$$(Name \sqcap Price \sqcap \exists Supplier.(OnlineAddress \sqcap Rating)). \tag{5}$$

After first rewriting Q as a projection of an index scan "$\pi_{Pd}(\sigma_C(\mathcal{K} :: Price))$" (see below), the system returns the result of evaluating Q on the description base. Thus, when browsing the page containing Q, a user sees an ordered list of inexpensive digital camera information ordered by price, and with each item displaying the name and price of the camera together with a sublist of supplier URL addresses and ratings for that camera.

Algorithm 4: $\textsc{Reduce}(S, \mathcal{T}, Rp)$

```
$result \leftarrow \emptyset$
foreach $C_1 \in S$ do
    $minimal \leftarrow true$
    foreach $C_2 \in S - \{C_1\}$ do
        if $\mathcal{T} \models \exists Rp.C_2 \sqsubseteq \exists Rp.C_1$ and $\mathcal{T} \not\models \exists Rp.C_1 \sqsubseteq \exists Rp.C_2$ then
            $minimal \leftarrow false$
            break
    if $minimal$ then
        $result \leftarrow result \cup \{C_1\}$
return $result$
```

This example suggests a useful extension to projection descriptions that can be easily accommodated. In particular, adding the case "$Pd :: Od$" as an additional option in (4) would enable a useful refinement to (5) above:

$$(Name \sqcap Price \sqcap \exists Supplier.((OnlineAddress \sqcap Rating) :: Rating)).$$

Consequently, the user will see the sublist of supplier URL addresses and ratings for each camera in the order of the supplier rating.

Also, an algebraic approach to querying description bases lends itself nicely to problems in query rewriting (witness the rewriting on Q above). In particular, the following identities hold in our algebra:

$$\begin{aligned} \sigma_C(\textsc{Sort}_{Od}(Q)) &\equiv \textsc{Sort}_{Od}(\sigma_C(Q)), \\ \textsc{Sort}_{Od}(\textsc{Sort}_{Od'}(Q)) &\equiv \textsc{Sort}_{Od}(Q) \text{ and} \\ \sigma_{C_1}(\sigma_{C_2}(Q)) &\equiv \sigma_{C_1 \sqcap C_2}(Q). \end{aligned}$$

The identities yield a normal form "$\textsc{Sort}_{Od_2}(\sigma_C(\mathcal{K} :: Od_1))$" for queries. One of the objectives of earlier work [3, 4] was to determine when queries in normal form could be evaluated efficiently, in particular when they could be rewritten as an index scan "$\sigma_C(\mathcal{K} :: Od_1)$" in which pruning made possible by the filtering concept C during an inorder traversal of the description index $\mathcal{K} :: Od_1$ was sufficient to guarantee logarithmic performance (and allowing that the choice of permutation for a concept list could be more limited than required by the original query). An underlying problem addressed in the earlier work is to determine the truth of a "$\prec_{\mathcal{T},C}$" predicate over the pair (Od_1, Od_2) of ordering descriptions, that is, to reason when, for terminology $\mathcal{T}$, the partial order induced by Od_1 contains the partial order induced by Od_2 over concepts that are subsumed by C. This predicate can be usefully "overloaded" to apply to projection descriptions as well. In particular, a sufficient condition for the additional algebraic identity

$$\pi_{Pd_2}(\pi_{Pd_1}(\sigma_C(Q))) \equiv \pi_{Pd_2}(\sigma_C(Q))$$

might be expressed as "$Pd_1 \prec_{\mathcal{T},C} Pd_2$", another topic for future work.

References

1. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
2. Franz Baader, Baris Sertkaya, and Anni-Yasmin Turhan. Computing the least common subsumer w.r.t. a background terminology. *J. Applied Logic*, 5(3):392–420, 2007.
3. Jeffrey Pound, Lubomir Stanchev, David Toman, and Grant Weddell. On Ordering Descriptions in a Description Logic. *20th International Workshop on Description Logics*, pages 123–134, 2007.
4. Jeffrey Pound, Lubomir Stanchev, David Toman, and Grant Weddell. On Ordering and Indexing Metadata for the Semantic Web. *21st International Workshop on Description Logics*, 2008.