On Ordering Descriptions in a Description Logic

Jeffrey Pound, Lubomir Stanchev, David Toman, and Grant Weddell

David R. Cheriton School of Computer Science
University of Waterloo, Canada

Abstract. We introduce a description language for specifying partial ordering relations over concept descriptions in description logics, and show how the language can be used in combination with binary trees to efficiently search a database that corresponds to a finite set of concept descriptions. The language consists of a pair of *ordering constructors* that support a form of exogenous indexing in which search criteria is independent of data, and a form of endogenous indexing in which the data itself provides search criteria. Our language can be refined in the same way as a description logic in that greater expressiveness and consequent richer search capability is achieved by adding additional ordering constructors.

1 Introduction

In earlier work, Stanchev and Weddell have shown how description logics can play a role in searching among objects in an object-oriented database [1]. In this paper, we introduce a description language deriving from their notion of an *extended index* that can be used to specify what we call *ordering descriptions*. Such descriptions correspond to strict partial orders over concept descriptions in a given description logic, in this paper $\mathcal{ALCQ}(\mathcal{D})$. We also show how an ordering description can be used in combination with binary trees to efficiently query a database of $\mathcal{ALCQ}(\mathcal{D})$ concept descriptions.

To illustrate, consider the case of an online supplier of photography equipment. As part of a web presence, the supplier maintains a binary tree of (descriptions of) items available for purchase, and maintains the tree in such a way that a traversal of the tree will visit items in a sequence compatible with a partial order defined by the following ordering description:

$$\mathit{ProductCode} : \mathrm{SaleItem}(\mathit{DisPrice} : \mathrm{Un}, \mathit{RegPrice} : \mathrm{Un}).$$

Intuitively, the ordering description specifies a non-descending major sort on the concrete feature *ProductCode*. Items having the same value for this feature then appear in two consecutive groups. The first group consists of items on sale that in turn occur in non-descending order of their discount price. The second group consists of items not on sale that in turn occur in non-descending order of their regular price. The supplier also maintains a terminology that includes the following constraints:

$$\begin{aligned} \text{SaleItem} &\sqsubseteq (\geq\ 3\ \mathit{Suppliers}\ \top), \\ \text{SaleItem} &\sqsubseteq (\mathit{Price} = \mathit{DisPrice}), \\ \neg\text{SaleItem} &\sqsubseteq (\mathit{Price} = \mathit{RegPrice}), \text{ and} \\ \top &\sqsubseteq (\mathit{DisPrice} < \mathit{RegPrice}). \end{aligned}$$

Now consider a request by an online user for *all descriptions of single-sourced digital cameras, retrieved in non-descending order of their price.* This query can be captured in our formalism by a concept description/ordering description pair as follows:

$$\langle \mathit{ProductCode} = \text{“digicam”} \sqcap \neg(\geq\ 2\ \mathit{Suppliers}\ \top),\ \mathit{Price} : \text{Un} \rangle.$$

Our results enable a procedure to reason that the tree maintained by the supplier refines the sort order for the query. To paraphrase, the query can be evaluated by an in-order search of the tree only, during which items will be returned in the order requested by the user. However, to avoid sorting, items will need to satisfy a property called *descriptive sufficiency* that relates to the above ordering description for the search tree, e.g., that each item description “supplies” a value for *ProductCode*. Note that this property will also make it possible to perform arbitrary rotations to ensure that the tree is balanced following the insertion of a new item.

Suppose the supplier decides that being single-sourced is a necessary condition for an item to not be on sale. Suppose in particular that the supplier adds the following constraint to the terminology:

$$\neg\text{SaleItem} \sqsubseteq \neg(\geq\ 2\ \mathit{Suppliers}\ \top).$$

Our results enable a further procedure to reason that the tree maintained by the supplier will now *support* the above query. This means not only that the same in-order search of the tree will suffice, but also that the number of subsumption checks in $\mathcal{ALCQ}(\mathcal{D})$ will be bounded both by the size of the result and by the logarithm (base two) of the number of items occurring in the tree.

The example illustrates the use of the two kinds of *ordering constructors* that make up our initial version of this ordering language. The separation of sale and non-sale items is an example of exogenous indexing in which search criteria is independent of data, while the major sort on *ProductCode* and minor independent sorts on *DisPrice* and *RegPrice* are examples of endogenous indexing in which the data itself provides search criteria. However, the language can be refined in the same way as a description logic in that greater expressiveness and consequent richer search capability becomes possible by adding additional ordering constructors. We suggest examples in our summary comments. Also, the exogenous constructor imposes no conditions on the selection of a particular dialect of description logic, while the endogenous constructor requires only that the dialect has a linearly ordered concrete domain.

The remainder of the paper is organized as follows. A formal definition of our ordering language, a refinement relationship among ordering descriptions, and a study of some of their properties follows in Section 2. We also include a Ref procedure for reasoning about refinement relationships. Although not known to be

a complete reasoner at this time, REF is easily able to recognize the above example case regarding sort order. In Section 3, we show how ordering descriptions define pruning criteria during search in a binary tree of concept descriptions. To ensure efficiency, such criteria will depend on the above-mentioned notion of descriptive sufficiency for concept descriptions occurring in the tree. Our main results complete the section in which we characterize *supported queries* for a given ordering condition and terminology. A summary and discussion follow in Section 4.

2 Ordering Descriptions

Our language for specifying partial orders over concept descriptions depends on the choice of underlying description logic. In this paper, we use $\mathcal{ALCQ}(\mathcal{D})$, a dialect that satisfies our illustrative requirements. However, our results apply to any choice of description logic that includes a linearly-ordered concrete domain.

Definition 1 (Description Logic $\mathcal{ALCQ}(\mathcal{D})$) *Let* $\{C, C_1, \ldots\}$, $\{R, S, \ldots\}$, $\{f, g, \ldots\}$ *and* $\{k, k_1, \ldots\}$ *denote sets of* primitive concept names, roles, concrete features, *and* constants *respectively. A* concept description *is then defined by the grammar:*

$$D, E \quad ::= \quad f < g \mid f < k \mid C \mid D \sqcap E \mid \neg D \mid (\geq\ n\ R\ D).$$

An inclusion dependency *is an expression of the form* $D \sqsubseteq E$. *A* terminology $\mathcal{T}$ *is a finite set of inclusion dependencies.*

An interpretation $\mathcal{I}$ *is a 3-tuple* $\langle \Delta_I, \Delta_C, \cdot^{\mathcal{I}} \rangle$ *where* Δ_I *is an arbitrary abstract domain,* Δ_C *a linearly ordered concrete domain, and* $\cdot^{\mathcal{I}}$ *an interpretation function that maps each concrete feature* f *to a total function* $f^{\mathcal{I}} : \Delta_I \rightarrow \Delta_C$, *each role* R *to a relation* $R^{\mathcal{I}} \subseteq \Delta_I \times \Delta_I$, *each primitive concept* C *to a set* $C^{\mathcal{I}} \subseteq \Delta_I$, *the* $<$ *symbol to the binary relation for the linear order on* Δ_C, *and* k *to a constant in* Δ_C. *The interpretation function is extended to arbitrary concepts in the standard way.*

An interpretation $\mathcal{I}$ *satisfies an inclusion dependency* $D \sqsubseteq E$ *if* $(D)^{\mathcal{I}} \subseteq (E)^{\mathcal{I}}$. $\mathcal{T} \models D \sqsubseteq E$ *if* $(D)^{\mathcal{I}} \subseteq (E)^{\mathcal{I}}$ *for all interpretations* $\mathcal{I}$ *that satisfy all inclusion dependencies in* $\mathcal{T}$.

For the remainder of the paper, we also use standard abbreviations, e.g., $D \sqcup E$ for $\neg(\neg D \sqcap \neg E)$, as well as the derived comparisons $\leq, >, \geq$, and $=$ on the concrete domain.

Notation 2 We write D^* to denote a description obtained from D by replacing all features f by f^*, roles R by R^*, and concepts C by C^*, and extend this notation in the obvious way to apply to inclusion dependencies and terminologies.

A formal definition of our description language for specifying partial orders over concept descriptions in $\mathcal{ALCQ}(\mathcal{D})$ now follows. As our introductory comments illustrate, we use the language in the next section for two purposes:

1. to define the relative positions of descriptions occurring in a search tree, and
2. as part of a query specifying the order in which such descriptions are to be presented to a user.

Definition 3 (Ordering Description) *Let D be an ALCQ(D) concept description, and f a concrete feature. An* ordering description *is defined by the grammar:*

$$Od \quad ::= \quad \text{Un} \;\mid\; f : Od \;\mid\; D(Od, Od).$$

An instance of the first (resp. second and third) production is called the null ordering *(resp.* feature value ordering *and* description ordering*).*

For a given terminology $\mathcal{T}$ and concept descriptions D and E, D is ordered before *E* by ordering description *Od* with respect to *$\mathcal{T}$, denoted $(Od)_{\mathcal{T}}(D, E)$, if $\mathcal{T} \not\models D \sqsubseteq \bot$, $\mathcal{T} \not\models E \sqsubseteq \bot$, and at least one of the following conditions holds:*

- *$Od =$ "$f : Od_1$" and $(\mathcal{T} \cup \mathcal{T}^*) \models (D \sqcap E^*) \sqsubseteq (f < f^*)$,*
- *$Od =$ "$f : Od_1$", $(Od_1)_{\mathcal{T}}(D, E)$ and $(\mathcal{T} \cup \mathcal{T}^*) \models (D \sqcap E^*) \sqsubseteq (f = f^*)$,*
- *$Od =$ "$D'(Od_1, Od_2)$", $\mathcal{T} \models D \sqsubseteq D'$ and $\mathcal{T} \models E \sqsubseteq \neg D'$,*
- *$Od =$ "$D'(Od_1, Od_2)$", $(Od_1)_{\mathcal{T}}(D, E)$ and $\mathcal{T} \models (D \sqcup E) \sqsubseteq D'$, or*
- *$Od =$ "$D'(Od_1, Od_2)$", $(Od_2)_{\mathcal{T}}(D, E)$ and $\mathcal{T} \models (D \sqcup E) \sqsubseteq \neg D'$.*

Two descriptions D and E are said to be incomparable with respect to an ordering *Od* and terminology *$\mathcal{T}$ if $\neg(Od)_{\mathcal{T}}(D, E)$ and $\neg(Od)_{\mathcal{T}}(E, D)$, or simply* incomparable *when Od and $\mathcal{T}$ are clear from context.*

Note that the null ordering denotes an unspecified or unknown ordering, and is used to capture circumstances when no (possibly residual) ordering relationship between descriptions is either sensible or needed.

Important properties of ordering descriptions are given by the following lemma.

Lemma 4 *For any terminology $\mathcal{T}$, ordering description Od, and concept descriptions D_1, D_2, and D_3:*

1. *If $(Od)_{\mathcal{T}}(D_1, D_2)$, then $\neg(Od)_{\mathcal{T}}(D_2, D_1)$;*
2. *If $(Od)_{\mathcal{T}}(D_1, D_2)$ and $(Od)_{\mathcal{T}}(D_2, D_3)$, then $(Od)_{\mathcal{T}}(D_1, D_3)$;*
3. *If $(Od)_{\mathcal{T}}(D_1, D_2)$, then $\mathcal{T} \models (D_1 \sqcap D_2) \sqsubseteq \bot$;*
4. *If $(Od)_{\mathcal{T}}(D_1, D_2)$, $\mathcal{T} \models D_3 \sqsubseteq D_1$ and $\mathcal{T} \not\models D_3 \sqsubseteq \bot$, then $(Od)_{\mathcal{T}}(D_3, D_2)$; and*
5. *If $(Od)_{\mathcal{T}}(D_1, D_2)$, $\mathcal{T} \models D_3 \sqsubseteq D_2$ and $\mathcal{T} \not\models D_3 \sqsubseteq \bot$, then $(Od)_{\mathcal{T}}(D_1, D_3)$.*

Properties 1 and 2 establish the basic requirement that any ordering description will define a strict (or irreflexive) partial order over descriptions in $\mathcal{ALCQ}(\mathcal{D})$. It turns out that these properties are sufficient conditions for pruning subtrees during search.

We now introduce the notion of *order refinement* that can be used, for example, to characterize order optimization—to formally define when sorting can be avoided when evaluating a query. Some properties of order refinement and an outline of procedures for reasoning about order refinement then follow.

Definition 5 (Order Refinement) *Assume a given terminology $\mathcal{T}$, concept description D and pair of ordering descriptions Od_1 and Od_2. Then, Od_1* refines *Od_2 with respect to $\mathcal{T}$ and D, written $Od_1 \prec_{\mathcal{T},D} Od_2$, if, for all concept descriptions E_1 and E_2 such that $\mathcal{T} \models (E_1 \sqcup E_2) \sqsubseteq D$:*

$$(Od_2)_{\mathcal{T}}(E_1, E_2) \;\; \textit{implies} \;\; (Od_1)_{\mathcal{T}}(E_1, E_2).$$

Od_1 is equivalent to Od_2 with respect to $\mathcal{T}$ and D, written $Od_1 \approx_{\mathcal{T},D} Od_2$, when $Od_1 \prec_{\mathcal{T},D} Od_2$ and $Od_2 \prec_{\mathcal{T},D} Od_1$. In all cases, D is called a parameter description.

Consider again an application in order optimization. One can, for example, avoid sorting a query result if the order in which the descriptions are retrieved is the same as the sort order specified by the query. Our notion of order refinement, however, is much more general, and the procedures that are outlined below for reasoning about order refinement can easily handle the case given in our introductory comments.

To further illustrate some basic capabilities of these procedures, assume that the order in which descriptions occur in a binary tree are defined by the ordering description "$f : g : \mathrm{Un}$", in particular, that an in-order traversal of the tree satisfies a major sort on concrete feature f and a minor sort on concrete feature g. Also assume that a user submits a query with the sort order "$f : \mathrm{Un}$". The procedures will deduce an order refinement between these ordering descriptions. Should the query also stipulate that any retrieved description should be subsumed by the concept description "$f = 27$", then the procedures will also deduce an order refinement between "$f : g : \mathrm{Un}$" and "$g : \mathrm{Un}$". Further details on query evaluation will be given in Section 3. Also note that an ability to reason about order refinement will have may related applications, e.g., in query optimization [2] and in index selection [1].

The following lemma establishes a number of equivalence properties of ordering descriptions that are independent of parameter descriptions.

Lemma 6 *For any terminology $\mathcal{T}$, ordering descriptions Od_1, Od_2, and Od_3 and concept descriptions D_1 and D_2:*

1. *If $\mathcal{T} \models D_2 \sqsubseteq D_1$, then $(D_2(Od_1, D_1(Od_2, Od_3))) \approx_{\mathcal{T},\top} (D_1(D_2(Od_1, Od_2), Od_3))$;*
2. *If $\mathcal{T} \models D_1 \sqsubseteq D_2$, then $(D_1(D_2(Od_1, Od_2), Od_3)) \approx_{\mathcal{T},\top} (D_1(Od_1, Od_3))$;*
3. *If $\mathcal{T} \models D_1 \sqsubseteq \neg D_2$, then $(D_1(D_2(Od_1, Od_2), Od_3)) \approx_{\mathcal{T},\top} (D_1(Od_2, Od_3))$;*
4. *If $\mathcal{T} \models D_2 \sqsubseteq D_1$, then $(D_1(Od_1, D_2(Od_2, Od_3))) \approx_{\mathcal{T},\top} (D_1(Od_1, Od_3))$;*
5. *If $\mathcal{T} \models \neg D_1 \sqsubseteq D_2$, then $(D_1(Od_1, D_2(Od_2, Od_3))) \approx_{\mathcal{T},\top} (D_1(Od_1, Od_2))$;*
6. *If $\mathcal{T} \models D \sqsubseteq (f < k)$ and $\mathcal{T} \models \neg D \sqsubseteq \neg(f < k)$, for some $k \in \Delta_C$, then $f : D(Od_1, Od_2) \approx_{\mathcal{T},\top} D(f : Od_1, f : Od_2)$; and*
7. *If $\mathcal{T} \models (f = g)$ and $Od_3 =$ "$g : Od_2$", then $f : Od_1 \approx_{\mathcal{T},\top} f : Od_1[Od_3/Od_2]$ for any occurrence of Od_3 in Od_1.*

Observe that the first is an associativity condition for nested instances of the description ordering constructor. This allows balancing a large number of occurrences of this constructor that might hypothetically comprise a (very large) ordering description.

$\textsc{Ref}\ (Od, \mathrm{Un}, \mathcal{T}, D) = \mathit{true}.$

$\textsc{Ref}\ (\mathrm{Un}, g : Od, \mathcal{T}, D) = (\mathcal{T} \models D \sqsubseteq (g = k) \text{ for some } k, \text{ and } \textsc{Ref}\ (\mathrm{Un}, Od, \mathcal{T}, D)).$

$\textsc{Ref}\ (\mathrm{Un}, D'(Od_1, Od_2), \mathcal{T}, D) = (\mathcal{T} \models D \sqsubseteq D' \text{ and } \textsc{Ref}\ (\mathrm{Un}, Od_1, \mathcal{T}, D)) \text{ or}$
$(\mathcal{T} \models D \sqsubseteq \neg D' \text{ and } \textsc{Ref}\ (\mathrm{Un}, Od_2, \mathcal{T}, D)).$

$\textsc{Ref}\ (f : Od', g : Od, \mathcal{T}, D) = (\mathcal{T} \models D \sqsubseteq (f = g) \text{ and } \textsc{Ref}\ (Od', Od, \mathcal{T}, D)) \text{ or}$
$(\mathcal{T} \models D \sqsubseteq (f = k) \text{ for some } k, \text{ and } \textsc{Ref}\ (Od', g : Od, \mathcal{T}, D)) \text{ or}$
$(\mathcal{T} \models D \sqsubseteq (g = k) \text{ for some } k, \text{ and } \textsc{Ref}\ (f : Od', Od, \mathcal{T}, D)).$

$\textsc{Ref}\ (f : Od, D'(Od_1, Od_2), \mathcal{T}, D) =$
$(\mathcal{T} \models D \sqsubseteq (f = k) \text{ for some } k, \text{ and } \textsc{Ref}\ (Od, D'(Od_1, Od_2), \mathcal{T}, D)) \text{ or}$
$(\mathcal{T} \models D \sqsubseteq \neg D' \text{ and } \textsc{Ref}\ (f : Od, Od_2, \mathcal{T}, D)) \text{ or}$
$(\mathcal{T} \models D \sqsubseteq D' \text{ and } \textsc{Ref}\ (f : Od, Od_1, \mathcal{T}, D)).$

$\textsc{Ref}\ (D'(Od_1, Od_2), g : Od, \mathcal{T}, D) =$
$(\mathcal{T} \models D \sqsubseteq (g = k) \text{ for some } k, \text{ and } \textsc{Ref}\ (D'(Od_1, Od_2), Od, \mathcal{T}, D)) \text{ or}$
$(\mathcal{T} \models D \sqsubseteq \neg D' \text{ and } \textsc{Ref}\ (Od_2, g : Od, \mathcal{T}, D)) \text{ or}$
$(\mathcal{T} \models D \sqsubseteq D' \text{ and } \textsc{Ref}\ (Od_1, g : Od, \mathcal{T}, D)).$

$\textsc{Ref}\ (D_1(Od_1, Od_2), D_2(Od_3, Od_4), \mathcal{T}, D) =$
$(\mathcal{T} \models D \sqsubseteq D_2 \text{ and } \textsc{Ref}\ (D_1(Od_1, Od_2), Od_3, \mathcal{T}, D)) \text{ or}$
$(\mathcal{T} \models D \sqsubseteq \neg D_2 \text{ and } \textsc{Ref}\ (D_1(Od_1, Od_2), Od_4, \mathcal{T}, D)) \text{ or}$
$(\mathcal{T} \models D \sqsubseteq D_1 \text{ and } \textsc{Ref}\ (Od_1, D_2(Od_3, Od_4), \mathcal{T}, D)) \text{ or}$
$(\mathcal{T} \models D \sqsubseteq \neg D_1 \text{ and } \textsc{Ref}\ (Od_2, D_2(Od_3, Od_4), \mathcal{T}, D)) \text{ or}$
$(\mathcal{T} \models (D \sqcap D_1) \equiv (D \sqcap D_2) \text{ and}$
$\textsc{Ref}\ (Od_1, Od_3, \mathcal{T}, D \sqcap D_1 \sqcap D_2) \text{ and}$
$\textsc{Ref}\ (Od_2, Od_4, \mathcal{T}, D \sqcap \neg D_1 \sqcap \neg D_2)).$

Fig. 1. An approximate structural refinement procedure.

It is possible to define a canonical form for ordering descriptions based on the above properties only, and to devise an effective procedure for computing this form, say $\textsc{Can}(Od, \mathcal{T})$, by a careful search for constants k, by orienting the equations for the first five properties, and so on. To determine if $Od_1 \prec_{\mathcal{T},D} Od_2$, we present the sound structural algorithm $\textsc{Ref}\ (Od_1, Od_2, \mathcal{T}, D)$ in Figure 1.

Lemma 7 *For any terminology $\mathcal{T}$, ordering descriptions Od_1 and Od_2, and concept description D, it follows that $Od_1 \prec_{\mathcal{T},D} Od_2$ holds if*

$$\textsc{Ref}\ (\textsc{Can}(Od_1, \mathcal{T}), \textsc{Can}(Od_2, \mathcal{T}), \mathcal{T}, D).$$

3 Indexing Descriptions

Our goal in this section is to show how ordering descriptions can be used to efficiently search an index consisting of a finite collection of concept descriptions in $\mathcal{ALCQ}(\mathcal{D})$. We begin with a formal definition of an underlying tree for an index.

Definition 8 (Description Tree) *Let D denote an arbitrary concept description in $\mathcal{ALCQ}(\mathcal{D})$. A* description tree *is an ordered rooted binary tree conforming to the grammar:*

$$Tr, L, R \quad ::= \quad \langle\rangle \quad | \quad \langle D, L, R\rangle.$$

An instance of the first production denotes an empty tree, while an instance of the second production denotes a node at the root of a tree with left subtree L, right subtree R, and labelled *by D. We write $\langle D, L, R\rangle \in Tr$ if $\langle D, L, R\rangle$ is a node occurring in Tr, and call any tree of the form $\langle D, \langle\rangle, \langle\rangle\rangle$ a* leaf *node.*

A description tree Tr is well formed for ordering description Od with respect to terminology $\mathcal{T}$ *if, for all $\langle D, L, R\rangle \in Tr$,*

- $\mathcal{T} \not\models D \sqsubseteq \bot$,
- $\neg(Od)_{\mathcal{T}}(D, D')$ *for all* $\langle D', L', R'\rangle \in L$, *and*
- $\neg(Od)_{\mathcal{T}}(D', D)$ *for all* $\langle D', L', R'\rangle \in R$.

When Od and $\mathcal{T}$ are clear from context, we say simply that Tr is well formed.

For a given ordering description Od, the conditions for Tr to be well formed provide the invariants for insertions of new nodes. For example, when inserting a new node for description D' in description tree $\langle D, L, R\rangle$, a new leaf node $\langle D', \langle\rangle, \langle\rangle\rangle$ must be added in subtree L if $(Od)_{\mathcal{T}}(D', D)$.

Definition 9 (Description Index) *Let $\mathcal{T}$ be a terminology, Od an ordering description, and Tr a well formed description tree with respect to Od and $\mathcal{T}$. A* description index *is a 3-tuple $\langle Tr, Od, \mathcal{T}\rangle$.*

We consider queries Q of the form $\langle D_Q, Od_Q\rangle$, where D_Q is a concept description in $\mathcal{ALCQ}(\mathcal{D})$ and Od_Q is an ordering description. A user presumes that query Q is evaluated with respect to an index $\langle Tr, Od, \mathcal{T}\rangle$ by first finding all concept descriptions E_i labelling nodes in Tr for which $\mathcal{T} \models E_i \sqsubseteq D_Q$ and then sorting the descriptions E_i according to Od_Q. In concrete terms, the first operation is accomplished by standard in-order tree traversal algorithms, assuming the following pruning conditions for a subtree $\langle E_i, L, R\rangle$ in Tr:

1. prune L if $(Od)_{\mathcal{T}}(E_i, D_Q)$, and
2. prune R if $(Od)_{\mathcal{T}}(D_Q, E_i)$.

If $\mathcal{T} \models E_i \sqsubseteq D_Q$ then E_i is included in the query result.

The correctness of this procedure is a simple consequence of the following lemma showing that no query result can exist in a pruned subtree. Note that its proof is a straightforward consequence of Lemma 4.

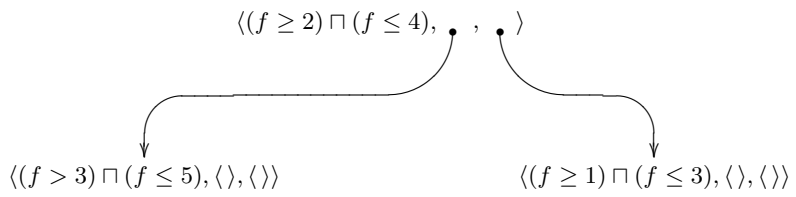

Fig. 2. A well formed description tree with respect to "f : Un".

Lemma 10 *For any description index $\langle Tr, Od, \mathcal{T}\rangle$, node $\langle D, L, R\rangle \in Tr$, and concept description E:*

1. $(Od)_{\mathcal{T}}(E, D)$ *implies* $\mathcal{T} \not\models D' \sqsubseteq E$ *for any node* $\langle D', L', R'\rangle \in R$, *and*
2. $(Od)_{\mathcal{T}}(D, E)$ *implies* $\mathcal{T} \not\models D' \sqsubseteq E$ *for any node* $\langle D', L', R'\rangle \in L$.

Unfortunately, the conditions for a description tree to be well formed are not strong enough to prevent a concept description labelling a Tr node to be ordered by Od prior to a concept description labelling a previous node in an in-order traversal of Tr. Thus, there is no guarantee that the descriptions returned during an in-order traversal will satisfy the ordering description for the index. Worse, rotations to ensure balance in Tr will not in general be possible. The following example illustrates these problems.

Example 11 Consider the description index $\langle Tr, f : \text{Un}, \emptyset\rangle$ in which Tr consists of the three nodes illustrated in Figure 2. Note that Tr is well formed since each description is satisfiable and since the root node is incomparable to both of the child nodes. If one considers a query that retrieves all descriptions, then descriptions will be retrieved out of order by an in-order traversal since the right child compares *left* of the left child with respect to f : Un.

Now consider what happens when a user submits a query Q for which D_Q is the same as the description labelling the right child, and when Tr itself is rotated right to make the left child the new root node. In this new circumstance, the above procedure for evaluating a query, in particular the strategy for pruning, is now incorrect in the sense that the description labelling the rightmost node will not be returned. This problem is caused by the rotation producing a description tree that is not well formed.

One way to overcome these problems is to introduce limitations on the concept descriptions that can label nodes in a description tree.

Definition 12 (Descriptive Sufficiency) *A concept description D is* sufficiently descriptive for ordering description Od with respect to terminology $\mathcal{T}$, *written $SD_{\mathcal{T}}(D, Od)$, if at least one of the following conditions hold:*

- Od = "Un",

– $Od =$ "$f : Od_1$", $SD_{\mathcal{T}}(D, Od_1)$, *and* $\mathcal{T} \models D \sqsubseteq (f = k)$,
– $Od =$ "$D'(Od_1, Od_2)$", $SD_{\mathcal{T}}(D, Od_1)$, *and* $\mathcal{T} \models D \sqsubseteq D'$, *or*
– $Od =$ "$D'(Od_1, Od_2)$", $SD_{\mathcal{T}}(D, Od_2)$, *and* $\mathcal{T} \models D \sqsubseteq \neg D'$,

for some $k \in \Delta_C$. *When* Od *and* $\mathcal{T}$ *are clear from context, we say simply that* D *is sufficiently descriptive.*

Again, it is possible to devise an effective procedure for deciding if $SD_{\mathcal{T}}(D, Od)$ holds, primarily by reusing the careful search for constants k presumed by the CAN procedure mentioned at the end of the previous section.

Descriptive sufficiency now enables us to say when rotations can be used to balance a description tree Tr.

Lemma 13 *Let* Od *be an ordering description and* $\mathcal{T}$ *a terminology. If concept descriptions* D_1 *and* D_2 *are sufficiently descriptive, then, for any description trees* Tr_1, Tr_2, *and* Tr_3 *that are well formed,* $\langle D_1, \langle D_2, Tr_1, Tr_2\rangle, Tr_3\rangle$ *is well formed if and only if* $\langle D_2, Tr_1, \langle D_1, Tr_2, Tr_3\rangle\rangle$ *is well formed.*

Lemma 14 that follows defines the additional properties of ordering descriptions that are needed to ensure the above procedure for searching a description index will return descriptions in non-descending order of the ordering description for the index. A proof of this is now a simple consequence of this lemma together with Lemma 4. As we show in the remainder of this section, it also becomes possible to ensure that query evaluation can be accomplished very efficiently in terms of the number of calls to a DL reasoner.

Lemma 14 *Let* Od *be an ordering description and* $\mathcal{T}$ *a terminology. For any concept description* D *and pair of concept descriptions* E_1 *and* E_2 *that are incomparable and sufficiently descriptive:*

1. *If* $(Od)_{\mathcal{T}}(D, E_1)$, *then* $(Od)_{\mathcal{T}}(D, E_2)$; *and*
2. *If* $(Od)_{\mathcal{T}}(E_1, D)$, *then* $(Od)_{\mathcal{T}}(E_2, D)$.

Definition 15 (Order Preserving Description Index) *An index* $\langle Tr, Od, \mathcal{T}\rangle$ *is* order preserving *if any concept description labelling any node in* Tr *is sufficiently descriptive.*

To summarize, adding a new concept description D to an order preserving index $\langle Tr, Od, \mathcal{T}\rangle$ is only possible if D is sufficiently descriptive, and is accomplished by adding $\langle D, \langle\rangle, \langle\rangle\rangle$ as a new leaf node in Tr and then performing rotations to ensure balance. If $\langle D', L, R\rangle \in Tr$, then the new node must be inserted in L when $(Od)_{\mathcal{T}}(D, D')$ holds, in R if $(Od)_{\mathcal{T}}(D', D)$ holds, and in either L or R otherwise.

Our main result now follows in which we characterize a number of the standard cases of queries for which an initial search followed by an index scan (and without a subsequent sort) will suffice to efficiently evaluate the query.

Definition 16 (Supported Query) *A description D is* sufficiently selective for ordering description Od with respect to terminology $\mathcal{T}$, *denoted $SS_{\mathcal{T}}(D, Od)$, if at least one of the following conditions hold:*

- $\mathcal{T} \models \top \sqsubseteq D$,
- $Od =$ "$f : Od_1$", $SS_{\mathcal{T}}(E, Od_1)$ *and* $\mathcal{T} \models D \equiv ((f = k) \sqcap E)$,
- $Od =$ "$f : Od_1$" *and* $\mathcal{T} \models D \equiv (f < k)$,
- $Od =$ "$f : Od_1$" *and* $\mathcal{T} \models D \equiv \neg(f < k)$,
- $Od =$ "$f : Od_1$" *and* $\mathcal{T} \models D \equiv (\neg(f < k) \sqcap (f < k'))$,
- $Od =$ "$D'(Od_1, Od_2)$", $SS_{\mathcal{T}}(E, Od_1)$ *and* $\mathcal{T} \models D \equiv (D' \sqcap E)$,
- $Od =$ "$D'(Od_1, Od_2)$", $SS_{\mathcal{T}}(E, Od_2)$ *and* $\mathcal{T} \models D \equiv (\neg D' \sqcap E)$, *or*
- $\mathcal{T} \models D \equiv (E \sqcup E')$, $SS_{\mathcal{T}}(E, Od)$ *and* $SS_{\mathcal{T}}(E', Od)$,

for some constants k and k' in Δ_C and descriptions E and E'.

A query $\langle D_Q, Od_Q \rangle$ is supported *by an order preserving description index $\langle Tr, Od', \mathcal{T} \rangle$ if and only if D_Q is sufficiently selective for ordering description Od' with respect to $\mathcal{T}$, and $Od' \prec_{\mathcal{T}, D_Q} Od_Q$.*

A procedure for deciding if the concept description of a query is sufficiently selective is an open problem at this time. However, an approximate procedure easily capable of recognizing our introductory example is straightforward, e.g., one that uses the above-mentioned careful search for constants k and k' to recognize the range query cases.

Theorem 17 *Let $\langle Tr, Od, \mathcal{T} \rangle$ be an order preserving description index and Q a supported query. Then Q can be evaluated in $O(log(n) + k)$ subsumption tests of $\mathcal{ALCQ}(\mathcal{D})$, where n is the number of nodes in Tr and k is the number of descriptions in the result.*

4 Summary and Discussion

We have proposed a language for specifying partial orders over concept descriptions that consists of an initial selection of two ordering constructors. The first is called feature value ordering, and is the standard notion of ordering supported by SQL and relational databases. The second is called description ordering, and can be viewed as a way of capturing indexing based on grid file techniques in which the focus is on organizing the data space in which data resides [3]. Multidimensional indices such as quad trees [4] are examples.

There are also a number of other ordering constructors that one might consider. Two possibilities for additional endogenous indexing are given by the following productions for our ordering language.

$$Od \quad ::= \quad f \ \mathbf{desc} : Od \ \mid \ \sqsubseteq$$

The first is an obvious extension that would enable (sub)orders satisfying non-ascending values for concrete features in descriptions, while the second appeals directly to subsumption checking in a description logic. More formally, the necessary revision to Definition 3 requires adding three conditions:

- $Od =$ "f **desc** : Od_1" and $(\mathcal{T} \cup \mathcal{T}^*) \models (D \sqcap E^*) \sqsubseteq (f^* < f)$,
- $Od =$ "f **desc** : Od_1", $(Od_1)_{\mathcal{T}}(D, E)$ and $(\mathcal{T} \cup \mathcal{T}^*) \models (D \sqcap E^*) \sqsubseteq (f = f^*)$, or
- $Od =$ "$\sqsubseteq$" and $\mathcal{T} \models D \sqsubseteq E$.

For the first constructor, it is also straightforward to extend proofs for Lemmas 4 and 10, and to extend the definition of description sufficiency in a way that will preserve Lemmas 13 and 14, which will then allow arbitrary rotations in a description tree and the possibility of removing a sort operator from a query plan, respectively. The same is not true, however, for the second constructor. In this case, Property 5 of Lemma 4 and Property 2 of Lemma 10 will no longer hold. Thus, pruning during search will only remain possible for right subtrees in a description index. There is also no obvious way to repair the definition of descriptive sufficiency in a way that will also preserve Lemmas 13 and 14.

We have demonstrated how our ordering language can support retrieving descriptions in response to queries. In an algebraic sense, the query language we have considered is very simple, consisting only of a selection operation for finding descriptions subsumed by a given "selection" concept, and a sort operation for ordering the set of descriptions produced by selection. DL systems such as Racer [5] that implement ABox reasoning already support the first of these operations. We believe our results provide some guidance on how DL systems can incorporate better support for sorting, for order optimization in ABox querying, and for ABox indexing.

There are several open problems and many possible avenues of further research. Finding a complete Ref procedure and either adding further operators to our query language or a sort capability to existing query languages such as EQL [6] are respective examples. Finally, an expanded version of this paper containing complete proofs is available as a technical report [7].

Acknowledgments

We thank Peter Tarle and Nortel for many valuable discussions on this work and for financial support. We also acknowledge the financial support in part by grants from NSERC of Canada.

References

1. Stanchev, L., Weddell, G.: Index Selection for Embedded Control Applications using Description Logics. In: Description Logics 2003, CEUR-WS vol.81 (2003) 9–18
2. Simmen, D.E., Shekita, E.J., Malkemus, T.: Fundamental Techniques for Order Optimization. In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data. (1996) 57–67
3. Nievergelt, J., Hinterberger, H., Sevcik, K.C.: The grid file: An adaptable, symmetric multikey file structure. ACM Trans. Database Syst. **9**(1) (1984) 38–71
4. Samet, H.: The quadtree and related hierarchical data structures. ACM Comput. Surv. **16**(2) (1984) 187–260
5. Volker Haarslev and Ralf Moller: Racer system description. In: International Joint Conference on Automated Reasoning, IJCAR. (2001)
6. Diego Calvanese and Giuseppe De Giacomo and Domenico Lembo and Maurizio Lenzerini and Riccardo Rosati: Epistemic First-Order Queries over Description Logic Knowledge Bases. In: Description Logics 2006, CEUR-WS vol.189 (2006)
7. Pound, J., Stanchev, L., Toman, D., Weddell, G.: On ordering descriptions in a description logic. Technical Report CS-2007-16, David R. Cheriton School of Computer Science, University of Waterloo (2007)