Towards a Theory of Search Queries

GEORGE H. L. FLETCHER
Eindhoven University of Technology
JAN VAN DEN BUSSCHE
Hasselt University and Transnational University of Limburg
DIRK VAN GUCHT
Indiana University
and
STIJN VANSUMMEREN
Université Libre de Bruxelles

The need to manage diverse information sources has triggered the rise of very loosely structured data models, known as dataspace models. Such information management systems must allow querying in simple ways, mostly by a form of searching. Motivated by these developments, we propose a theory of search queries in a general model of dataspaces. In this model, a dataspace is a collection of data objects, where each data object is a collection of data items. Basic search queries are expressed using filters on data items, following the basic model of Boolean search in information retrieval. We characterize semantically the class of queries that can be expressed by searching. We apply our theory to classical relational databases, where we connect search queries to the known class of fully generic queries, and to dataspaces where data items are formed by attribute-value pairs. We also extend our theory to a more powerful, associative form of searching, where one can ask for objects that are similar to objects satisfying given search conditions. Such associative search queries are shown to correspond to a very limited kind of joins. We show that the basic search language extended with associative search can exactly define the queries definable in a restricted fragment of the semijoin algebra working on an explicit relational representation of the dataspace.

This research was done while S. Vansummeren was a Postdoctoral Fellow of the Research Foundation-Flanders, at Hasselt University, Belgium.
Authors' addresses: G. Fletcher, Faculty of Mathematics and Computer Science, Eindhoven University of Technology, Postbus 513, 5600 MB Eindhoven, The Netherlands; email: g.h.l.fletcher@tue.nl; J. Van den Bussche, Database and Theoretical Computer Science Group, Hasselt University and Transnational University of Limburg, Agoralaan D, B-3590 Diepenbeek, Belgium; email: jan.vandenbussche@uhasselt.be; D. Van Gucht, Computer Science Department, Indiana University, Lindley Hall 215, 150 S. Woodlawn Avenue, Bloomington, IN 47405; email: vgucht@cs.indiana.edu; S. Vansummeren, Laboratory for Web and Information Technology, Université Libre de Bruxelles, avenue F.D. Roosevelt 50, CP 165/15, B-1050 Brussels, Belgium; email: stijn.vansummeren@ulb.ac.be.

DOI 10.1145/1862919.1862925 http://doi.acm.org/10.1145/1862919.1862925

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Modal logic*; H.2.1 [**Database Management**]: Logical Design—*Data models*; H.2.3 [**Database Management**]: Languages—*Query languages*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms: Design, Languages, Theory

Additional Key Words and Phrases: Dataspaces, genericity, search

ACM Reference Format:

Fletcher, G. H. L., Van den Bussche, J., Van Gucht, D., and Vansummeren, S. 2010. Towards a theory of search queries. ACM Trans. Datab. Syst., 35, 4, Article 28 (November 2010), 33 pages. DOI = 10.1145/1862919.1862925 http://doi.acm.org/10.1145/1862919.1862925

1. INTRODUCTION

One can say that querying by a form of searching has become the norm. Indeed, in most current information systems such as Web search engines, e-commerce sites, or desktop search systems, as well as in classical information retrieval systems such as library catalogs or document repositories, users query the database by means of a search interface. Searching is expressed by means of keywords that can be combined with Boolean operators. Such a basic search facility is much weaker than the standard database query languages, where select–project–join queries are considered the minimum, and full-fledged first-order logic is considered the norm (cf. Codd's relational algebra and calculus). Contemporary languages such as SQL/PSM or XQuery are even computationally complete.

Database queries are a major theme of database theory [Abiteboul et al. 1995]. In general, queries are generic mappings from databases to relations, where "generic" refers to invariance under isomorphisms [Chandra and Harel 1980; Aho and Ullman 1979]. Many classes of database queries have been identified and characterized in terms of semantic properties; expressibility in various query languages; or computability under various complexity limitations. Since searching is such a simple, natural, and important form of querying, it appears that search queries deserve to have their own chapter in the theory of database queries. Our goal in this article is to propose a first draft of such a chapter. Apart from the foundational motivation, our work is further motivated by two important trends in data management research: dataspaces and usability.

Dataspaces [Halevy et al. 2006; Dong and Halevy 2007; Dittrich and Vaz Salles 2006] are a new type of databases characterized by a very loosely structured data model and geared towards the management of data coming from a diverse set of sources. Dataspaces are typically queried by a form of searching. In essence, the data in a dataspace is modeled as a collection of objects, where each object is a collection of attribute-value pairs. Dataspaces are queried by searching for conditions on attributes or values, or by following links between objects. Relationally complete querying of dataspace-like databases was studied earlierin the context of models for data integration [Jain et al.

1995]. Recent work on data processing for cloud computing has also focused on dataspace-like databases (e.g., DeCandia et al. [2007] and Olston et al. [2008]).

Improving the usability of database systems has been an issue pretty much since the beginning of database research, as witnessed for example, by the past research on universal relation interfaces (surveyed by Ullman [1989, Chapter 17]). Recently, Jagadish et al. [2007] have revived our interest in this topic; he argues, among other things, that queries involving explicit joins or subqueries are too cumbersome to express, and stimulates us to ask how far we can get with simpler forms of querying such as searching, or with more implicit or automatic ways of joining information.

So, in a nutshell, in this article, we try to formally understand the question of how much database querying can be done using a basic search language, possibly extended with a simple facility for following links between objects.

We should also clarify what we do not do in this article. We do not investigate how searching can be implemented efficiently. Rather, we focus on expressive power. Also, because of this focus, we ignore other important issues that have been investigated in research on keyword search in relational, tree-structured (XML), and semistructured graph databases [Golenberg et al. 2008; Qin et al. 2009; we give just two recent references]. The two main such issues are automatically finding connections among objects in the database that contain the given keywords (which is a nice approach to the usability question), and ranking the results of a keyword search.

Concretely, the contributions of this article can be summarized as follows.

(1) We define a general formal model of dataspaces, where a dataspace is a collection of data objects, and where an object is a collection of data items. On these data items, we assume a number of abstract filter predicates to be defined. These filters naturally serve as atomic search conditions: searching a dataspace with a filter returns all objects containing an item satisfying the filter. We obtain a basic search language by combining atomic searches using the Boolean set operators union, intersection, and difference.

(2) Search queries are defined in general as functions on dataspaces that map a dataspace to one of its subsets. The question then arises of exactly which such mappings are definable in the basic search language. This question turns out to be naturally answered by requiring the search queries to be invariant under a natural indistinguishability relation on objects. The concept of genericity (invariance) has always been a central theme in the development of the theory of database queries [Abiteboul et al. 1995]; we have tried here to get at the right genericity concept for search queries.

(3) We apply this semantic characterization to the case of classical database relations. A relation can be viewed as a dataspace by viewing the tuples as sets of attribute-value pairs. When the classical selection conditions "attribute equals constant" are used as filters on such attribute-value pairs, we obtain as basic search queries precisely the class of search queries that are "fully generic" in the sense of Beeri et al. [1996, 1997]. (We hasten to add that these authors looked at full genericity of queries in

a complex-object setting much richer than mere search queries on flat relations.)

(4) We extend the basic model to allow for so-called associative search, where one can search not just for all objects satisfying some search query, but also for all objects that are somehow linked to those objects. This extension amounts to adding a link operator to the basic search language, much in the spirit of modal logic [Blackburn et al. 2001, 2007]. Associative search is indeed a feature of most dataspace and keyword search query languages proposed in the literature [Halevy et al. 2006; Dong and Halevy 2007; Dittrich and Vaz Salles 2006; Golenberg et al. 2008]. Actually, in these proposals, links between objects are often assumed to be found automatically by the system. Since in this article we are interested in a detailed analysis of expressive power, we focus on the case where the linking conditions are explicitly specified in the query.

(5) A question we find interesting is how search query languages relate to standard database query languages. Those languages remain relevant in the search and dataspace setting. For example, SPARQL [W3C 2008], the standard language to query RDF graphs [W3C 2004], is closely related to database queries on ternary relations [Fletcher 2008; Gutierrez et al. 2004; Pérez et al. 2009]; RDF graphs can be viewed as a dataspace model.

Under the natural assumption that linking between two objects is done in terms of a set of similarity relations among the items in these objects, we can indeed relate associative search to standard database query languages. Specifically, we observe that associative search queries are definable in the semijoin algebra: the version of the relational algebra where the join operator is replaced by the semijoin [Leinders et al. 2005; Leinders and Van den Bussche 2007; Ross and Janevski 2005]. Here, the semijoin algebra works on the natural representation of a dataspace as a binary relation; abstract filters are used as selection conditions, and abstract similarity predicates are used as join conditions.

Conversely however, not every semijoin query is an associative search query. We actually identify three kinds of constructions that are definable in the semijoin algebra but not in our associative search language, and prove that the fragment of the semijoin algebra in which these three constructions are disallowed fully characterizes our associative search queries.

(6) Finally, we show that our general abstract theory is workable by applying it to the most common concrete dataspace setting, where data items are attribute-value pairs. Unlike the application to classical relations mentioned earlier, here there is no fixed schema and objects can have multiple values for the same attribute. We consider a natural repertoire of filters that can test if the attribute and the value, equals, or is different from, a finite number of possibilities. In this setting, the semantic characterization of basic search queries can be rephrased in a very intuitive manner, as we will show. We will also instantiate the semijoin algebra characterization of associative search queries to the attribute-value setting. Associative search queries over attribute-value dataspaces are thus characterized as

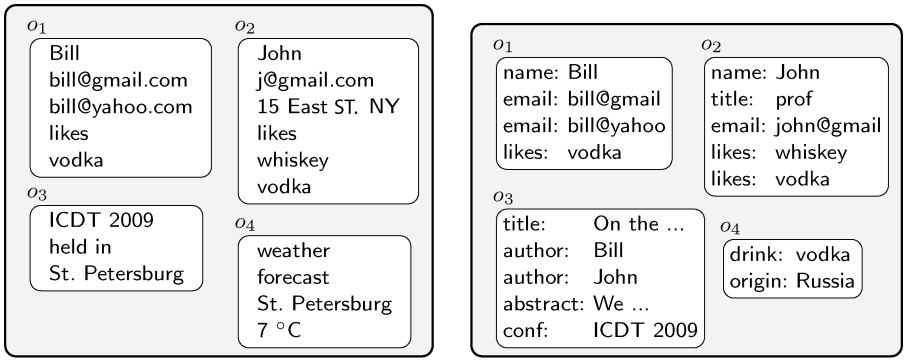


Fig. 1. (a) An example document space, and (b) an example attribute-value space.

the queries expressible in a simple and attractive fragment of the semijoin algebra; this time, dataspaces are represented by ternary relations with schema (Oid, Att, Val), selection conditions are constant equalities (on attributes as well as on values; remember that there is no dataspace schema), and all semijoins are simple equijoins.

The current article extends earlier work on this topic, including nontrivial proofs of results, applications of the theory, and further discussion [Fletcher et al. 2009].

This article is organized as follows. Section 2 develops the general theory of search queries on the model of abstract dataspaces, both in terms of Boolean search queries and associative search queries. Section 3 presents the application to classical database relations. Section 4 presents the application to attribute-value dataspaces. We finish with a discussion in Section 5.

2. A THEORY OF SEARCH QUERIES

In this section we propose a theory of search queries in a general model of dataspaces. In this model, a *dataspace* is a collection of data *objects*. Each data object is a collection of data elements called *items*, drawn from a universe of items $\mathcal{I}$. Although we will keep $\mathcal{I}$ abstract in this section in order to obtain a generally applicable theory, a rich set of well-known, practical data models can be obtained by picking concrete interpretations for $\mathcal{I}$. For instance:

— Items could be words over some alphabet. Objects then correspond to documents in the classical Boolean model of information retrieval [Manning et al. 2008, chapter 1], and dataspaces correspond to collections of such documents. Figure 1 illustrates such a dataspace, consisting of documents concerning events in St. Petersburg.

— Items could be pairs (a, v) of attributes a and values v. Objects, being finite sets of such pairs, then intuitively describe the attributes of a real-world entity. (Note that there can be multiple values for the same attribute.) A dataspace is just a collection of such descriptions. Figure 1, for example,

depicts an attribute-value dataspace describing researchers, drinks, and papers.

— Staying with attribute-value pairs, we could consider those dataspaces whose objects contain only attributes from a fixed relation schema, and contain a single value for each such attribute. Since such objects correspond to traditional relational tuples, such dataspaces, having these tuples as objects, are classical database relations.

Definition 1 (*Dataspaces*). Formally, let $\mathcal{I}$ be a set of data elements called items. An *object over* $\mathcal{I}$ (by short: $\mathcal{I}$*-object*) is a finite, nonempty set of items. A *dataspace over* $\mathcal{I}$ (by short: $\mathcal{I}$*-dataspace*) is a finite set of $\mathcal{I}$-objects.

Throughout the remainder of this section we fix $\mathcal{I}$ arbitrarily and simply talk about objects and dataspaces instead of $\mathcal{I}$-objects and $\mathcal{I}$-dataspaces, respectively. For ease of exposition, however, we will illustrate our definitions and results in the situation where $\mathcal{I}$ is the set of all words, as in classical information retrieval. The other concrete situations shown here, where $\mathcal{I}$ is a set of attribute-value pairs, are studied in greater depth in Sections 3 and 4.

2.1 Boolean Search

As a first basic language to express search queries on dataspaces, we consider the Boolean Search Language BSL. Similar to the Boolean information retrieval setting, searches in BSL are formed by combining atomic search conditions (keywords) using the set operators union, intersection, and difference. First, let us make the notion of a keyword precise.

Definition 2 (*Keywords*). For our purposes, a *keyword* is defined semantically as a subset $k \subseteq \mathcal{I}$. An item i is said to *match* k if it is an element $i \in k$. An object o *satisfies* k, denoted $o \models k$, if it contains an item that matches k, that is, if $o \cap k$ is nonempty.

Keywords naturally serve as atomic search conditions: searching a dataspace with a keyword returns all objects that satisfy the keyword. For example, let contains i denote the keyword that consists of all items that contain i (e.g., as a substring). Searching by contains ICDT then returns all objects in the dataspace that contain an item in which ICDT occurs as a substring.

In what follows, we write $\mathcal{K}$ for the set of all keywords. Two kinds of keywords deserve special attention:

Definition 3 (*Wildcard and literal keywords*). The universe $\mathcal{I}$ itself is the keyword that matches all items. We refer to it as the *wildcard* in what follows. To draw attention to the fact that it is used as a keyword rather than the universe of all items, we will denote it by $\star$ instead of $\mathcal{I}$.

For every item i, the singleton keyword $\{i\}$ is called a *literal* keyword. We often abuse notation and omit the set braces from these keywords, writing i instead of $\{i\}$.

Equipped with the atomic search conditions, we are ready to formally define the Boolean Search Language BSL.

Definition 4. The expressions of BSL are given by the grammar:

$$e \ ::= \ k \mid e \text{ and } e \mid e \text{ or } e \mid e \text{ except } e,$$

where k ranges over the keywords in $\mathcal{K}$. Semantically, an expression e can be applied to a dataspace D, resulting in a subset $e(D)$ of D defined as follows.

$$\begin{aligned} k(D) &:= \{o \in D \mid o \models k\} \\ (e_1 \text{ and } e_2)(D) &:= e_1(D) \cap e_2(D) \\ (e_1 \text{ or } e_2)(D) &:= e_1(D) \cup e_2(D) \\ (e_1 \text{ except } e_2)(D) &:= e_1(D) - e_2(D). \end{aligned}$$

Note that the language is a bit redundant, as e_1 and e_2 is equivalent to e_1 except (e_1 except e_2).

It is important to appreciate the existential nature of these semantics. Thus, the expression k_1 except k_2 does not return all objects that contain an item that matches k_1 but not k_2; rather, it returns all objects that contain an item that matches k_1, but do not contain an item that matches k_2.

Example 5. To illustrate the semantics, consider the setting where items are words. Then the expression:

$$e_1 := \{\text{weather}\} \text{ and } \{\text{forecast}\} \text{ and } \{\text{St. Petersburg}\},$$

retrieves all documents in which the words weather, forecast, and St. Petersburg all occur. By our convention that we identify items i with the singleton keyword $\{i\}$, which matches i only, we will simply write this expression as:

$$\text{weather and forecast and St. Petersburg},$$

in what follows. As another example, suppose that, for each item i, similar to i is a keyword that matches all items similar to i according to some notion of similarity. Then the expression:

$$e_2 := \text{vodka except (similar to whiskey)},$$

retrieves all documents in which the drink vodka occurs, but no drink similar to whiskey occurs. In particular, on the document space in Figure 1, e_1 and e_2 return $\{o_4\}$ and $\{o_1\}$, respectively.

Of course, in real-life situations, one rarely has all keywords available to search with. For that reason, we denote by BSL(K) the fragment of BSL expressions that use only keywords in $K \subseteq \mathcal{K}$. Thus, e_1 in Example 5 is in BSL($\{\{\text{weather}\}, \{\text{forecast}\}, \{\text{St. Petersburg}\}\}$), but e_2 is not.

As just defined, every BSL expression defines a search query:

Definition 6 (*Search Query*). A *search query* is a mapping q from dataspaces to dataspaces such that $q(D) \subseteq D$ for each dataspace D. We say that a search query is *definable in* BSL if there exists an expression e such that $e(D) = q(D)$ for every D.

Of course not all search queries are definable in BSL. Which ones are? We can answer this question by identifying three typical properties of BSL queries:

additivity, K-safety, and K-distinguishing. We define these three properties next.

Definition 7 (*Additivity, K-safety, K-distinguishing*). Let q be a search query and let $K \subseteq \mathcal{K}$ be a set of keywords.

— We say that q is *additive* if:

$$q(D) = \bigcup_{o \in D} q(\{o\}),$$

for any dataspace D.

— We say that q is *K-safe* if for any dataspace D and any $o \in q(D)$, we have that $o \models k$ for at least one $k \in K$.

— Two objects o_1 and o_2 are called *K-equivalent* if for all $k \in K$ we have $o_1 \models k$ if and only if $o_2 \models k$. We denote this by $o_1 \simeq_K o_2$. (Clearly, $\simeq_K$ is an equivalence relation.) We then say that q is *K-distinguishing* if for any two dataspaces D_1 and D_2 and objects $o_1 \in D_1$ and $o_2 \in D_2$ that are K-equivalent, we have $o_1 \in q(D_1)$ if and only if $o_2 \in q(D_2)$.

These three properties represent three distinctive features of search queries. Additivity simply states that the query can be processed one object at a time. K-safety states that we cannot retrieve arbitrary objects from the dataspace, but only objects that satisfy at least one of the specified keywords. This is also the case in all real-life search engines and information retrieval systems. Finally, K-distinguishing naturally states that the query can only distinguish between objects based on their satisfaction of specified keywords.

As a matter of fact, additivity already follows from K-distinguishing, since the latter property implies $o \in q(D)$ if and only if $o \in q(\{o\})$, which readily implies additivity. We stated the property of additivity separately because we will need it later independently of K-distinguishing.

We establish the following semantic characterization.

THEOREM 8. *Let $K \subseteq \mathcal{K}$ be a set of keywords. The following are equivalent for any search query q:*

(1) q is definable in $\mathsf{BSL}(K)$*;*

(2) q is L-safe and L-distinguishing, for some finite set $L \subseteq K$.

PROOF. $(1 \rightarrow 2)$ Suppose that $e \in \mathsf{BSL}(K)$ defines q. Take L to be the finite set of keywords mentioned in e. The properties of L-safety and L-distinguishing are then readily verified by structural induction on e (details omitted).

$(2 \rightarrow 1)$ Suppose that search query q is L-safe and L-distinguishing with $L \subseteq K$ finite. First observe that $\simeq_L$ has only a finite number of equivalence classes. Indeed, since two objects are L-equivalent only if they satisfy the same keywords in L, we can find for each equivalence class C, a subset of keywords $L_C \subseteq L$ such that:

$$C = \{o \mid o \models k \text{ for all } k \in L_C \text{ and } o \not\models k \text{ for all } k \in L - L_C\}.$$

Since there are only a finite number of subsets $L_C \subseteq L$, there can only be a finite number of equivalence classes.

Then let $C_1, \ldots, C_s$ be those equivalence classes of $\simeq_L$ that contain an object o with $o \in q(\{o\})$. We say that these equivalence classes *satisfy* q. Note that since q is L-distinguishing, for each object o' we have $o' \in q(\{o'\})$ if and only if $o' \in C_1 \cup \cdots \cup C_s$.

Then observe that each equivalence class $C \in \{C_1, \ldots, C_s\}$ is definable in $\mathsf{BSL}(L)$ in the sense that we have an expression e_C such that $o \in C$ if and only if, $o \in e_C(\{o\})$. Indeed, pick $o \in C$ arbitrarily, let $\{k_1, \ldots, k_m\} = L_C$ be the finite set of all keywords in L that o satisfies, and let $\{l_1, \ldots, l_n\} = L - L_C$ be the finite set of all keywords in L that o does not satisfy. Let:

$$e_C := (k_1 \text{ and } \ldots \text{ and } k_m) \text{ except } (l_1 \text{ or } \ldots \text{ or } l_m).$$

Clearly, for any object o' we have $o' \in e_C(\{o'\})$ if and only if $o' \simeq_L o$ (and hence $o' \in C$). Note that e_C is a valid expression only if $m \geq 1$. However, since q is L-safe and C satisfies q, we know that the objects in C must satisfy at least one keyword in L. Thus we are assured that $m \geq 1$.

Then q is defined in $\mathsf{BSL}(L)$ by $e := e_{C_1} \text{ or } \ldots \text{ or } e_{C_s}$. Indeed, by the only-if direction we know that every expression in $\mathsf{BSL}(L)$ is L-distinguishing, and thus additive. In particular, e is additive. Then observe that for any dataspace D and any $o \in D$ we have:

$$\begin{aligned}
& o \in (e_{C_1} \text{ or } \ldots \text{ or } e_{C_s})(D) \\
& \quad \Leftrightarrow o \in (e_{C_1} \text{ or } \ldots \text{ or } e_{C_s})(\{o\}) \quad \text{(since } e \text{ is additive)} \\
& \quad \Leftrightarrow o \in e_{C_1}(\{o\}) \cup \cdots \cup e_{C_s}(\{o\}) \\
& \quad \Leftrightarrow o \in C_1 \cup \cdots \cup C_s \quad \text{(by construction of } e_{C_i}) \\
& \quad \Leftrightarrow o \in q(\{o\}) \\
& \quad \Leftrightarrow o \in q(D) \quad \text{(since } q \text{ is } L\text{-distinguishing, and hence additive).}
\end{aligned}$$

As such, e defines q. □

Note that the finiteness of L is crucial in the proof.

We point out that in the presence of the wildcard keyword, the issue of L-safety becomes moot:

COROLLARY 9. *When $K \subseteq \mathcal{K}$ includes the wildcard keyword $\star$, a search query is definable in $\mathsf{BSL}(K)$ if and only if it is L-distinguishing for some finite $L \subseteq K$.*

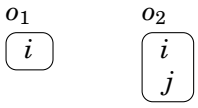
PROOF. The only-if direction follows directly from Theorem 8. For if-direction, first observe that $o \simeq_L o'$ if and only if, $o \simeq_{L \cup \{\star\}} o'$. Hence, since q is L-distinguishing, it is also $(L \cup \{\star\})$-distinguishing. Moreover, since all objects satisfy $\star$, *every* query is $(L \cup \{\star\})$-safe. Therefore, q is both $(L \cup \{\star\})$-safe and $(L \cup \{\star\})$-distinguishing, and hence definable in $\mathsf{BSL}(K)$ by Theorem 8. □

The following example gives two illustrations of how one can show that certain queries are not definable in BSL.

Example 10. One may wonder if negations of keywords can be expressed: can we express the query "$\neg k$",

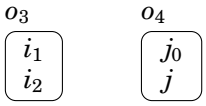
$$(\neg k)(D) = \{o \in D \mid \exists i \in o : i \notin k\},$$

which retrieves all objects containing an item that does not match some fixed keyword k? The answer in general is no. In proof, suppose that k is a keyword with at least one item $i \in k$ and at least one item $j \notin k$. Then the query $\neg k$ is not L-distinguishing for any $L \subseteq \{k, \star\}$. Indeed, consider the following objects:



Then $o_1 \simeq_L o_2$ for any such L, yet o_2 satisfies the query whereas o_1 does not. Hence, $\neg k$ is not definable in $\mathsf{BSL}(k, \star)$.

Of course, one can add "negated keywords" (complements of keywords relative to $\mathcal{I}$) directly to the set of allowed keywords to express such queries. One may then wonder whether that is enough to already allow all Boolean combinations of keywords to be expressed. For example, can we now retrieve all objects containing an item that matches neither k_1 nor k_2? The answer is still no. In proof, suppose that k_1 and k_2 are two keywords and i_1, i_2, j_0, and j are items such that i_1 matches k_1 but not k_2; i_2 matches k_2 but not k_1; j_0 matches neither k_1 nor k_2; and j matches both k_1 and k_2. Let $\neg k_1$ and $\neg k_2$ be the negated keywords of k_1 and k_2, respectively. ($\neg k_1 := \mathcal{I} - k_1$, and similarly for $\neg k_2$.) Then the query "$\neg k_1 \wedge \neg k_2$" is not L-distinguishing for any $L \subseteq \{k_1, k_2, \neg k_1, \neg k_2, \star\}$ and is thus not definable in $\mathsf{BSL}(k_1, k_2, \neg k_1, \neg k_2, \star)$. Indeed, consider the objects:



Then $o_3 \simeq_L o_4$ for any such L, but o_4 satisfies the query, whereas o_3 does not.

We conclude this section by pointing out that Theorem 8 continues to hold if one is only interested in definability of q with respect to a particular class of dataspaces, in the following sense.

Definition 11. Let $\mathcal{D}$ be a subclass of the class of all dataspaces and let $K \subseteq \mathcal{K}$ be a set of keywords.

— A search query q is *definable on $\mathcal{D}$ in* BSL if there exists an expression e such that $e(D) = q(D)$ for every $D \in \mathcal{D}$.

— A search query q is *K-safe on $\mathcal{D}$* if for any $D \in \mathcal{D}$ and any $o \in q(D)$, we have $o \models k$ for at least on $k \in K$.

— A search query q is *K-distinguishing on $\mathcal{D}$* if for any two dataspaces $D_1 \in \mathcal{D}$ and $D_2 \in \mathcal{D}$ and any two objects $o_1 \in D_1$ and $o_2 \in D_2$ that are K-equivalent, we have $o_1 \in q(D_1)$ if and only if $o_2 \in q(D_2)$.

COROLLARY 12. *Let $K \subseteq \mathcal{K}$ be a set of keywords. The following are equivalent for any search query q and any class of dataspaces $\mathcal{D}$.*

(1) q is definable on $\mathcal{D}$ in $\mathsf{BSL}(K)$.

(2) q is L-safe and L-distinguishing on $\mathcal{D}$, for some finite set $L \subseteq K$.

As in Corollary 9, the condition that q be L-safe can be dropped when $\star \in K$.

PROOF. (1 → 2) Suppose that $e \in \mathsf{BSL}(K)$ defines q on $\mathcal{D}$. Since $e \in \mathsf{BSL}(K)$, we know by Theorem 8 that the query defined by e is L-safe and L-distinguishing on the class of all dataspaces, for some finite $L \subseteq K$. This implies in particular that e is L-safe and L-distinguishing on $\mathcal{D}$. Hence, since q coincides with e on $\mathcal{D}$, so is q.

(2 → 1) Suppose that search query q is L-safe and L-distinguishing on $\mathcal{D}$, with $L \subseteq K$ finite. Define the search query p on the class of all dataspaces as follows.

$$p(D) = \{o \in D \mid \text{ there exists } D' \in \mathcal{D} \text{ and } o' \in q(D') \text{ with } o \simeq_L o'\}.$$

This search query is L-safe on the class of all dataspaces. Indeed, if $o \in p(D)$ then there exists some $D' \in \mathcal{D}$ and some $o' \in q(D')$ with $o \simeq_L o'$. Since q is L-safe on $\mathcal{D}$ we know that there exists $k \in L$ with $o' \models k$. Then, since $o \simeq_L o'$, also $o \models k$.

The query is also L-distinguishing on the class of all dataspaces: let D_1 and D_2 be arbitrary dataspaces, and suppose that $o_1 \in D_1$ and $o_2 \in D_2$ are L-equivalent. Then:

$$\begin{aligned} o_1 \in p(D_1) &\Leftrightarrow \text{ there exists } D' \in \mathcal{D} \text{ and } o' \in q(D') \text{ with } o_1 \simeq_L o' \\ &\Leftrightarrow \text{ there exists } D' \in \mathcal{D} \text{ and } o' \in q(D') \text{ with } o_2 \simeq_L o' \\ &\Leftrightarrow o_2 \in p(D_2). \end{aligned}$$

By Theorem 8, p is definable in $\mathsf{BSL}(L)$ on the class of all dataspaces. The corollary then follows since $p(D) = q(D)$ for all $D \in \mathcal{D}$. Indeed, if $D \in \mathcal{D}$ and $o \in q(D)$ then also $o \in p(D)$ by definition of p. Conversely, if $D \in \mathcal{D}$ and $o \in p(D)$ then there exists $D' \in \mathcal{D}$ and $o' \in q(D')$ with $o \simeq_L o'$. Since q is L-distinguishing on $\mathcal{D}$, also $o \in q(D)$. □

2.2 Associative Search

Since BSL queries are additive, BSL cannot define queries that relate objects to other objects. In other words, BSL cannot do joins. For example, in the information retrieval setting (where items are words) the search query:

> "retrieve all objects that share a word with an object in which ICDT 2009 occurs"

is not additive, and therefore not definable in BSL. In dataspace systems, however, we often want to be able to retrieve not just all objects that satisfy some search query, but also all objects that are related to those objects [Dong and Halevy 2007; Dittrich and Vaz Salles 2006]. To this end, we extend our theory as follows.

Definition 13 (*Link condition*). Let $\mathcal{O}$ denote the set of all objects. A *link condition* is a binary relationship $\lambda \subseteq \mathcal{O} \times \mathcal{O}$ on objects. We write $\lambda(o, o')$ to denote $(o, o') \in \lambda$, and write $\mathcal{L}$ for the set of all link conditions.

We then define the *associative search language* ASL as an extension of the basic search language BSL with an operator for retrieving related objects.

Definition 14. The expressions of ASL are given by the grammar

$$e \ ::= \ k \mid e \ \mathsf{and}\ e \mid e \ \mathsf{or}\ e \mid e \ \mathsf{except}\ e \mid \mathsf{link}\langle\lambda\rangle\, e,$$

where λ ranges over $\mathcal{L}$. The semantics of the new construct is given by:

$$(\mathsf{link}\langle\lambda\rangle\, e)(D) := \{o \in D \mid \exists o' \in e(D) : \lambda(o, o')\}.$$

Example 15. As a simple example on document spaces, let shares be the link condition that links o to o' if and only if o and o' share a common word. Then the expression link⟨shares⟩ ICDT 2009 returns all documents that share a word with a document in which ICDT 2009 occurs. In particular, on the dataspace in Figure 1, it returns $\{o_3, o_4\}$.

As another example, consider a document space of scientific biographies, and let advisor be the link condition that links o to o' if and only if o is the PhD advisor of o'. Then the expression link⟨advisor⟩(link⟨advisor⟩ St. Petersburg State University) returns the biographies of scientists who are the grand-advisors of scientists associated with St. Petersburg State University (who are the advisor of the advisor of a scientist whose biography mentions the university).

Of course, in real-life situations, we do not have all possible link conditions available to search with. For that reason, we are primarily interested in the queries definable in ASL(K, L), where ASL(K, L) denotes the fragment of ASL expressions that use only keywords in $K \subseteq \mathcal{K}$ and link conditions in $L \subseteq \mathcal{L}$.

It turns out that there is a connection between ASL and modal logics (see Blackburn et al. [2001] for a recent survey of modal logics). Indeed, ASL is a modal logic, in the sense that objects are "possible worlds/states", link conditions induce "transitions" between these states, and ASL is a navigational language for reasoning over such transition systems. We can make this connection precise by defining a bisimulation notion appropriate for ASL.

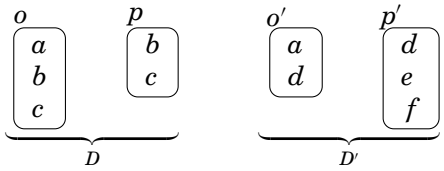
Definition 16 (*Bisimilarity*). A *pointed dataspace* is a pair (D, o) with D a dataspace and o an object in D. Let $K \subseteq \mathcal{K}$ and $L \subseteq \mathcal{L}$ be sets of keywords and link conditions, respectively. Two pointed dataspaces (D, o) and (D', o') are *n-bisimilar*, or *n-bisimulation equivalent*, under K and L, denoted $(D, o) \rightleftarrows_n^{K,L} (D', o')$, if $o \simeq_K o'$ and in addition the following conditions hold for $n > 0$:

Forth For any $\lambda \in L$, if $\lambda(o, p)$ for some $p \in D$, then there is some $p' \in D'$ such that $\lambda(o', p')$ and $(D, p) \rightleftarrows_{n-1}^{K,L} (D', p')$.

Back For any $\lambda \in L$, if $\lambda(o', p')$ for some $p' \in D'$, then there is some $p \in D$ such that $\lambda(o, p)$ and $(D, p) \rightleftarrows_{n-1}^{K,L} (D', p')$.

Observe in particular that for the base case $n = 0$, $(D, o) \rightleftarrows_0^{K,L} (D', o')$ if $o \simeq_K o'$.

Example 17. For example, if $K = \{\star, a\}$; $L = \{\mathsf{shares}\}$ (with shares as in Example 15); and D and D' are as follows:



then $(D, o) \rightleftarrows_n^{K,L} (D', o')$ and $(D, p) \rightleftarrows_n^{K,L} (D', p')$ for any n. Indeed, clearly $o \simeq_K o'$ and $p \simeq_K p'$. To verify the Forth property, shares links o in D only to o and p by which we can respond in D' by o' and p' respectively. Similarly, shares links p only to p and o, by which we can respond in D' by p' and o' respectively. The Back property is verified similarly. In contrast, $(D, o) \not\rightleftarrows_n^{K,L} (D', p')$ for any n since $o \not\simeq_K p'$.

Bisimilarity characterizations play a fundamental role in the study of the expressivity of modal logics, analogous to the classical Ehrenfeucht-Fraïssé characterizations employed in the study of the expressivity of first order logic [Goranko and Otto 2007]. Towards delimiting the expressive power of ASL, we next introduce a notion of bisimulation-invariance.

Definition 18. Let $\mathcal{D}$ be a class of dataspaces. A query q is $\rightleftarrows_n^{K,L}$*-invariant on* $\mathcal{D}$ if for any $(D, o) \rightleftarrows_n^{K,L} (D', o')$ with $D, D' \in \mathcal{D}$ we have $o \in q(D) \Leftrightarrow o' \in q(D')$. A query is $\rightleftarrows_n^{K,L}$*-invariant* if it is $\rightleftarrows_n^{K,L}$-invariant on the entire class of all dataspaces.

If we define the nesting depth of link operators in an expression e, denoted by $depth(e)$, as follows:

$$\begin{aligned} depth(k) &:= 0 \\ depth(e_1 \text{ and } e_2) &:= \max(depth(e_1), depth(e_2)) \\ depth(e_1 \text{ or } e_2) &:= \max(depth(e_1), depth(e_2)) \\ depth(e_1 \text{ except } e_2) &:= \max(depth(e_1), depth(e_2)) \\ depth(\text{link}\langle\lambda\rangle\, e) &:= 1 + depth(e) \end{aligned}$$

then adapting to our setting the known expressivity characterization from the model theory of modal logic [Goranko and Otto 2007, Theorem 32], we obtain the following helpful lemma. We give the details of our proof adaptation, as the tools developed therein are of independent interest.

LEMMA 19. *Let K be a finite nonempty set of keywords and let L be a finite nonempty set of link conditions. Then the following are equivalent for any search query q and any class $\mathcal{D}$ of dataspaces:*

(1) q is definable on $\mathcal{D}$ in ASL$(K \cup \{\star\}, L)$ *by an expression e with nesting depth of* link *operators at most n;*

(2) q is $\rightleftarrows_n^{K,L}$-invariant *on $\mathcal{D}$.*

PROOF. $(1 \to 2)$ Let e be an arbitrary expression in ASL$(K \cup \{\star\}, L)$ and let $n = depth(e)$ be the nesting depth of link constructs in e. It is readily verified by

induction on e that the query defined by e is $\rightleftarrows_n^{K,L}$-invariant on the class of all dataspaces, and therefore also $\rightleftarrows_n^{K,L}$-invariant on any subclass $\mathcal{D}$.

$(2 \rightarrow 1)$ The crucial step in proving the converse direction from (2) to (1) is the construction, for every pointed dataspace (D, o), of a so-called *characteristic expression* $e^n_{(D,o)}$ in $\mathsf{ASL}(K \cup \{\star\}, L)$, with the property that for all pointed dataspaces (D', o'),

$$o' \in e^n_{(D,o)}(D') \Leftrightarrow (D', o') \rightleftarrows_n^{K,L} (D, o). \tag{1}$$

This construction proceeds by induction on n, for all (D, o) simultaneously.

— When $n = 0$, $e^n_{(D,o)}$ is the $\mathsf{BSL}(K \cup \{\star\})$ expression:

$$e^0_{(D,o)} := (k \text{ and } \ldots \text{ and } k') \text{ except } (l \text{ or } \ldots \text{ or } l'),$$

where $\{k, \ldots, k'\}$ are all keywords in K satisfied by o, and $\{l, \ldots, l'\}$ are all keywords in K not satisfied by o. If o does not satisfy any keyword in K then $e^0_{(D,o)}$ is $\star$ except $(l \text{ or } \ldots \text{ or } l')$.

— Inductively, let $\mathsf{link}[\lambda]e$ abbreviate the expression $\star$ except $\mathsf{link}\langle\lambda\rangle(\star \text{ except } e)$ which returns all objects that are only linked by λ to those objects returned by e. Then $e^{n+1}_{(D,o)}$ is the $\mathsf{ASL}(K \cup \{\star\}, L)$ expression with link nesting depth $n+1$ given by:

$$e^{n+1}_{(D,o)} := e^0_{(D,o)} \text{ and } \bigwedge_{\lambda \in L} \Big(\underbrace{\bigwedge_{\substack{p \in D \\ \lambda(o,p)}} \mathsf{link}\langle\lambda\rangle\, e^n_{(D,p)}}_{\text{forth}} \wedge \underbrace{\mathsf{link}[\lambda] \bigvee_{\substack{p \in D \\ \lambda(o,p)}} e^n_{(D,p)}}_{\text{back}} \Big).$$

Intuitively, the subexpression $e^0_{(D,o)}$ verifies that $o' \simeq_K o$. The subexpression labeled "forth" verifies that for every $\lambda \in L$ and every $p \in D$ with $\lambda(o, p)$ for some $p \in D$ there exists $p' \in D'$ such that $\lambda(o', p')$ and $(D', p') \rightleftarrows_n^{K,L} (D, p)$. The subexpression labeled "back" verifies that o' does not have $p' \in D'$ with $\lambda(o', p')$ for which we cannot find a corresponding $p \in D$ with $\lambda(o, p)$ and $(D', p') \rightleftarrows_n^{K,L} (D, p)$.

As such, if $\{(D_1, o_1), (D_2, o_2), \ldots\}$ is the set of all pointed dataspaces (D, o) with $D \in \mathcal{D}$ and $o \in q(D)$, q is defined on $\mathcal{D}$ by the possibly infinite expression:

$$e^n_{(D_1,o_1)} \text{ or } e^n_{(D_2,o_2)} \text{ or } \ldots$$

Since K and L are finite, however, there are only a finite number of characteristic expressions $e^n_{(D,o)}$ up to logical equivalence. Therefore, this expression can always be made finite, as desired. □

Like Theorem 8 for BSL, Lemma 19 provides a tool to show certain queries are (un)definable in $\mathsf{ASL}(K, L)$ for various K and L—a tool that we will repeatedly employ in the following sections.

2.3 On Searching by Similarity Link Conditions and the Semijoin Algebra

In their full generality, link conditions, as binary relations, can be arbitrarily complex, and need not adhere to any reasonable notion of associative search. Consider, for instance, the link condition *subset* $:= \{(o, o') \mid o \subseteq o'\}$. Then

$\mathsf{link}\langle subset\rangle$ ICDT 2009 returns all objects o for which there exists some o' that mentions ICDT 2009 and $o \subseteq o'$. Such link conditions are known as a *set joins* in database theory [Leinders and Van den Bussche 2007; Sarawagi and Kirpal 2004], and have more to do with full-blown querying than searching.

Moreover, a flexible search query language should allow the link conditions to be expressed within the language itself.

For these reasons we next introduce a restricted set of link conditions, the simlinks, which intuitively link two objects o and o' by, (1) searching within o using a keyword k; (2) searching within o' using a keyword l; and (3) comparing the two search results and requiring that they contain a pair of similar items.

Definition 20. A *similarity relation* (or *simrel* for short) is a binary relation $\sim$ on items, $\sim \subseteq \mathcal{I} \times \mathcal{I}$. Let k and l be keywords and let $\sim$ be a simrel. Then the expression $k \sim l$ is called a *simlink* and can be used as a link condition with the following semantics:

$$k \sim l := \{(o, o') \in \mathcal{O} \times \mathcal{O} \mid \exists i \in o \cap k : \exists j \in o' \cap l : i \sim j\}.$$

Henceforth we write $\mathcal{S}$ for the set of all simrels.

Example 21. For a simple example on document spaces consider the simrel soviet between words such that w_1 soviet w_2 if w_1 is a location name (e.g., city, street, or building names) from the Soviet era, and w_2 is the corresponding post-Soviet name. For example, Leningrad soviet St. Petersburg pairs up objects mentioning the Soviet and post-Soviet names, respectively, of this well-known city. Then the expression:

$$\mathsf{link}\langle\star \text{ soviet } \star\rangle(\mathsf{ICDT}),$$

retrieves all documents containing Soviet versions of location names mentioned in documents about ICDT, and the expression:

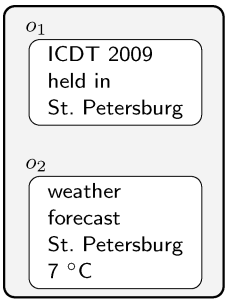
$$\mathsf{link}\langle\star \text{ soviet Peterhof}\rangle(\mathsf{ICDT}),$$

retrieves all documents containing the Soviet names of the Russian town Peterhof, if mentioned in documents about ICDT. In Section 4.2 we will see more examples of simlinks.

For the remainder of this article, we will always use the language ASL with simlinks as link conditions. In particular, if $K \subseteq \mathcal{K}$ and $S \subseteq \mathcal{S}$ then we write $\mathsf{ASL}(K, S)$ for the fragment of ASL expressions that use only keywords in K and link conditions in $\{k \sim l \mid k, l \in K \text{ and } \sim \in S\}$.

Relationship to the semijoin algebra Search queries are a special kind of database queries. It is therefore natural to ask how the language ASL (with simlinks) compares to more standard query languages. Observing that the link operator is a kind of semijoin, a comparison with the semijoin algebra seems a good approach to this question. (Recall that semijoin algebra is the version of the relation algebra where the join operator is replaced by the semijoin operator [Leinders et al. 2005; Leinders and Van den Bussche 2007].)

Since the relational algebra works on relations instead of dataspaces, we need a relational representation of a dataspace.



(a) Dataspace D

id	item
o_1	ICDT 2009
o_1	held in
o_1	St. Petersburg
o_2	weather
o_2	forecast
o_2	St. Petersburg
o_2	7 °C

(b) Binary relation $rep(D)$

Fig. 2. A dataspace and its relational representation.

$$\frac{}{T\colon \{\mathsf{id},\mathsf{item}\}} \qquad \frac{E\colon \Sigma \quad \mathsf{item} \in \Sigma \quad k \in \mathcal{K}}{\sigma_k(E)\colon \Sigma} \qquad \frac{E\colon \Sigma \quad \Delta \subseteq \Sigma}{\pi_\Delta(E)\colon \Delta} \qquad \frac{E_1\colon \Sigma \quad E_2\colon \Sigma}{E_1 \cup E_2\colon \Sigma}$$

$$\frac{E_1\colon \Sigma \quad E_2\colon \Sigma}{E_1 - E_2\colon \Sigma} \qquad \frac{E_1\colon \Sigma_1 \quad E_2\colon \Sigma_2 \quad \Sigma_1 \cap \Sigma_2 = \{\mathsf{id}\}}{E_1 \ltimes E_2\colon \Sigma_1}$$

$$\frac{E_1\colon \Sigma_1 \quad E_2\colon \Sigma_2 \quad \Sigma_1 \cap \Sigma_2 = \{\mathsf{item}\} \quad \sim \in \mathcal{S}}{E_1 \ltimes_\sim E_2\colon \Sigma_1} \qquad \frac{E_1\colon \{\mathsf{id},\mathsf{item}\} \quad E_2\colon \{\mathsf{id},\mathsf{item}\} \quad \sim \in \mathcal{S}}{E_1 \ltimes_{=,\sim} E_2\colon \{\mathsf{id},\mathsf{item}\}}$$

Fig. 3. Syntax of SA.

Definition 22. For each dataspace D let $rep(D)$ be the binary relation $\{(o, i) \mid o \in D \text{ and } i \in o\}$ over the relation schema {id, item}.

The objects in the id-column of this relation are regarded as object identifiers; the semijoin algebra will not be able to peek inside the objects in any other way than working with the item column. This is a standard representation for objects in the relational model (e.g., Agrawal et al. [2001] and Litwin et al. [1991]); note, however, that any reasonable representation would suffice in what follows. An example is shown in Figure 2.

In the version of the semijoin algebra we are using, keywords on items are used as selection conditions, and simrels on items and equality on objects (viewed as ids) are used as semijoin conditions. The relation symbol T (for "Table") stands for the binary representation of the input dataspace. Every expression of the algebra has an output schema that is either empty, the schema {id, item} itself, or one of the unary schemas {id} or {item} (we do not have renaming). Expressions must be well-typed, for example, we only allow the union of two relations over the same schema. The full syntax of semijoin algebra (abbreviated SA) expressions, together with the derivation of their output schemas, is given in Figure 3. There, the notation $E\colon \Sigma$ denotes that E is a legal expression with output schema Σ.

The semantics of projection π, union $\cup$, and difference $-$ is well known; the semantics of the semijoin operators is as follows.

$$\begin{aligned}
\sigma_k(R) &:= \{t \in R \mid t(\mathsf{item}) \in k\}\\
R_1 \ltimes R_2 &:= \{t_1 \in R_1 \mid \exists t_2 \in R_2 : t_2(\mathsf{id}) = t_1(\mathsf{id})\}\\
R_1 \ltimes_{\sim} R_2 &:= \{t_1 \in R_1 \mid \exists t_2 \in R_2 : t_1(\mathsf{item}) \sim t_2(\mathsf{item})\}\\
R_1 \ltimes_{=,\sim} R_2 &:= \{(o,i) \in R_1 \mid \exists j : (o,j) \in R_2 \text{ and } i \sim j\}.
\end{aligned}$$

So, $\ltimes$ is a normal natural semijoin on the common id attribute; $\ltimes_{\sim}$ is a $\sim$-semijoin on the common item attribute; and $\ltimes_{=,\sim}$ is a combination of the two.

Definition 23. A search query q *is definable in* SA if there exists an SA expression $E\colon \Sigma$ with $\mathsf{id} \in \Sigma$ such that $q(D) = \pi_{\mathsf{id}}(E)(rep(D))$, for all dataspaces D. We also say that E *defines* q in this case.

The following is now expected:

PROPOSITION 24. *Let $K \subseteq \mathcal{K}$ and $S \subseteq \mathcal{S}$. Every search query definable in* ASL(K, S) *is also definable in* SA(K, S).

PROOF. Here is a straight syntactic translation:

$$\begin{aligned}
\mathsf{SA}[k] &:= \sigma_k(T)\\
\mathsf{SA}[e_1 \text{ and } e_2] &:= \pi_{\mathsf{id}}\mathsf{SA}[e_1] - (\pi_{\mathsf{id}}\mathsf{SA}[e_1] - \pi_{\mathsf{id}}\mathsf{SA}[e_2])\\
\mathsf{SA}[e_1 \text{ or } e_2] &:= \pi_{\mathsf{id}}\mathsf{SA}[e_1] \cup \pi_{\mathsf{id}}\mathsf{SA}[e_2]\\
\mathsf{SA}[e_1 \text{ except } e_2] &:= \pi_{\mathsf{id}}\mathsf{SA}[e_1] - \pi_{\mathsf{id}}\mathsf{SA}[e_2]\\
\mathsf{SA}[\mathsf{link}\langle k \sim l\rangle\, e] &:= \pi_{\mathsf{id}}(\mathsf{SA}[k] \ltimes_{\sim} \pi_{\mathsf{item}}(\mathsf{SA}[l] \ltimes \pi_{\mathsf{id}}\mathsf{SA}[e]))
\end{aligned}$$ □

Is the converse true as well? The answer is no, and we will perform a thorough analysis of the situation, with the goal of arriving at a well-defined fragment of SA that is equivalent to ASL.

The first observation is that SA can express boolean combinations of keywords. Indeed, let $k \in \mathcal{K}$ be a keyword. The SA expression $\pi_{\mathsf{id}}(T - \sigma_k(T))$ defines the negated keyword $\neg k$, that is, the query $q(D) = \{o \in D \mid \exists i \in o : i \notin k\}$. We have already seen in Example 10 that negated keywords, and more generally, Boolean combinations of keywords, are not definable in ASL. This is easily repaired by closing the keywords under the Boolean operators.

Definition 25. For $K \subseteq \mathcal{K}$, let K^* be the smallest set of keywords containing K that is closed under the operators $\neg$ and $\vee$, where

$$\begin{aligned}
\neg k &:= \{i \in \mathcal{I} \mid i \notin k\} \text{ and}\\
k \vee l &:= \{i \in \mathcal{I} \mid i \in k \text{ or } i \in l\}.
\end{aligned}$$

Note that whenever K is nonempty, K^* necessarily includes the wildcard keyword $\star$, which matches all items, since $\star \equiv k \vee \neg k$.

We can now revise Proposition 24 as follows:

PROPOSITION 24 [REVISED] 1. *Let $K \subseteq \mathcal{K}$ and $S \subseteq \mathcal{S}$. Every search query definable in* ASL(K^*, S) *is also definable in* SA(K, S).

PROOF. The syntactic translation from the proof of Proposition 24 can be extended as follows: (where $k \in K$, and φ and ψ stand for Boolean combinations of keywords),

$$\begin{aligned} \mathsf{SA}[k] &:= \sigma_k(T) \\ \mathsf{SA}[\neg\varphi] &:= T - \mathsf{SA}[\varphi] \\ \mathsf{SA}[\varphi \vee \psi] &:= \mathsf{SA}[\varphi] \cup \mathsf{SA}[\psi]. \end{aligned}$$

□

The next proposition points at a number of distinct query constructions definable in SA but not in ASL.

PROPOSITION 26. *None of the following* SA *queries is definable in* ASL*, even over the Boolean closure of keywords*

$$\begin{aligned} E_1 &:= \pi_{\mathsf{id}}(\sigma_k(T) \ltimes_{=,\sim} \sigma_l(T)) \\ E_2 &:= \pi_{\mathsf{id}}(T - T \ltimes_{\sim} \pi_{\mathsf{item}}\sigma_k(T)) \\ E_3 &:= \pi_{\mathsf{id}}(\sigma_k(T) \ltimes_{\sim} \pi_{\mathsf{item}}(\sigma_l(T) \ltimes_{\sim} \pi_{\mathsf{item}}\sigma_m(T))) \\ E_4 &:= \pi_{\mathsf{id}}(T \ltimes_{\sim}(\pi_{\mathsf{item}}(T) - \pi_{\mathsf{item}}(T \ltimes \pi_{\mathsf{id}}\sigma_k(T)))). \end{aligned}$$

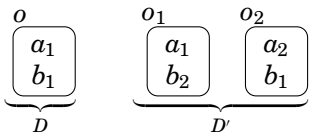
More precisely, for each of these queries we can pick the keywords k, l, m, and simrel $\sim$ *such that the query is not definable in* $\mathsf{ASL}(K^*, S)$ *with* $K = \{k, l, m\}$ *and* $S = \{\sim\}$.

Note that, referring to the items $i \in k$ as k-items, the four expressions, E1–E4, define the following queries.

(1) Retrieve all objects containing a k-item i and an l-item j such that $i \sim j$.
(2) Retrieve all objects containing an item that is not $\sim$ to any k-item in the dataspace.
(3) Retrieve all objects containing a k-item that is $\sim$ to some l-item j in the dataspace, and j itself is $\sim$ to some m-item in the dataspace.
(4) Retrieve all objects containing an item i that is $\sim$ to an item that is not present in any object containing a k-item.

These queries are shown to be undefinable in $\mathsf{ASL}(K^*, S)$ by showing that they are not bisimulation invariant, then invoking Lemma 19.

PROOF OF PROPOSITION 26. For the query q_1 expressed by E_1, we pick $k = \{a_1, a_2\}$ and $l = \{b_1, b_2\}$, with $a_1 \sim b_1$ and $a_2 \sim b_2$. Now consider:



Let $K = \{k, l\}$ and let L be the set of all link conditions $\{p \sim p' \mid p, p' \in K^*\}$. Then (D, o), (D', o_1), and (D', o_2) are all bisimilar (n-bisimilar for any n) under K^* and L. The verification of this claim proceeds by induction on n.

— Case $n = 0$. Here it suffices to show that $o \simeq_{K^*} o_1 \simeq_{K^*} o_2$, which readily follows from the observation that, writing $k \wedge l$ for $\neg(\neg k \vee \neg l)$,

$$K^* = \{k, l, \neg k, \neg l, k \vee l, k \vee \neg l, \neg k \vee l, k \wedge l, k \wedge \neg l, \neg k \wedge l, \emptyset, \star\}.$$

By construction, o, o_1, and o_2 all satisfy the same keywords in this set.

— Case $n > 0$. Suppose that $(D, o) \rightleftarrows_{n-1}^{K^*,L} (D', o_1) \rightleftarrows_{n-1}^{K^*,L} (D', o_2)$. We only need to show that $(D, o) \rightleftarrows_{n}^{K^*,L} (D', o_1)$ and $(D, o) \rightleftarrows_{n}^{K^*,L} (D', o_2)$, since $(D', o_1) \rightleftarrows_{n}^{K^*,L} (D', o_2)$ then follows by transitivity of $\rightleftarrows_{n}^{K^*,L}$. We check the Forth and Back properties.

(Forth) Let $\lambda = p \sim p'$ be an arbitrary simlink in L, and let o' be an arbitrary object in D with $\lambda(o, o')$. We need to show that there exists o'_1 and o'_2 in D' such that, (1) $\lambda(o_1, o'_1)$ and $(D, o) \rightleftarrows_{n-1}^{K^*,L} (D', o'_1)$; and (2) $\lambda(o_2, o'_2)$ and $(D, o) \rightleftarrows_{n-1}^{K^*,L} (D', o'_2)$. Theretoward, first observe that, since D is a singleton, we must have $o' = o$. Moreover, since the only items in o that are related by $\sim$ are a_1 and b_1, $\lambda(o, o)$ implies $a_1 \in p$, $b_1 \in p'$, and $a_1 \sim b_1$.

(1) Then, since $a_1 \in o_1$ and $a_2 \in o_2$, certainly also $\lambda(o_1, o_2)$. Furthermore, $(D, o) \rightleftarrows_{n-1}^{K^*,L} (D', o_2)$ by the induction hypothesis. As such, the Forth property holds for (D', o_1).

(2) To see that the Forth property also holds for (D', o_2), first observe that for any of the keywords in K^*, we have $a_1 \in p$ if and only if, $a_2 \in p$ and $b_1 \in p'$ if and only if, $b_2 \in p'$. Hence, since $a_1 \in p$ and $b_1 \in p'$, also $a_2 \in p$ and $b_2 \in p'$. Thus, $a_2 \in o_2 \cap p$, $b_2 \in o_1 \cap p$, and $a_2 \sim b_2$. Hence $\lambda(o_2, o_1)$. Furthermore, $(D, o) \rightleftarrows_{n-1}^{K^*,L} (D', o_1)$ by the induction hypothesis. As such, the Forth property holds for (D', o_2).

(Back) Similar to Forth.

So in particular, (D, o) and (D', o_1) are bisimilar under K^* and L. Yet, $o \in q_1(D)$, whereas $o_1 \notin q_1(D')$. Hence q_1 is not definable in $\mathsf{ASL}(K^*, S)$ by Lemma 19.

For the query q_2, expressed by E_2, we use items a, a', and b with $k = \{b\}$ and $a' \sim b$. Now consider $D = \{o_1, o_2\}$ and $D' = \{o'_1, o_2\}$ with $o_1 = \{a, a'\}$; $o_2 = \{b\}$; and $o'_1 = \{a'\}$. Let $K = \{k, l\}$ and let L be the set of all link conditions $\{p \sim p' \mid p, p' \in K^*\}$. Then (D, o_1) is bisimilar to (D', o'_1) under K^* and L (as can again be verified by induction), but $o_1 \in q_2(D)$, whereas $o'_1 \notin q_2(D')$.

For the query q_3 expressed by E_3, we pick $k = \{a\}$; $l = \{b, b_1, b_2\}$; and $m = \{c\}$, with $a \sim b$, $b \sim c$, $a \sim b_1$, and $b_2 \sim c$. Now consider $D = \{o_1, o_2\}$ and $D' = \{o_1, o_2\}$ with $o_1 = \{a\}$; $o_2 = \{b, c\}$; and $o'_2 = \{b_1, b_2, c\}$. Let $K = \{k, l, m\}$ and let L be the set of all link conditions $\{p \sim p' \mid p, p' \in K^*\}$. Then (D, o_1) is bisimilar to (D', o_1) under K^* and L, but $o_1 \in q_3(D)$ whereas $o_1 \notin q_3(D')$.

For the query q_4 expressed by E_4, we use items a, b, and c, with $k = \{a\}$, and $\sim$ interpreted as equality. Now consider $D = \{o_1, o_2\}$ and $D' = \{o'_1, o_2\}$ with $o_1 = \{b, c\}$; $o_2 = \{a, b\}$; and $o'_1 = \{b\}$. Let $K = \{k\}$ and let L be the set of all link conditions $\{p \sim p' \mid p, p' \in K^*\}$. Then (D, o_1) is bisimilar to (D', o'_1) under K^* and L, but $o_1 \in q_4(D)$ whereas $o'_1 \notin q_4(D')$. □

Proposition 26 can inspire us to restrict SA so as to obtain a fragment equivalent to ASL. E_1 suggests that we should banish the combined semijoin $\ltimes_{=,\sim}$.

E_2 suggests that we should not allow unrestricted use of the result of a $\ltimes_\sim$ semijoin; we should project the result. But E_3 shows we should not project on {item}, so we conclude we must always project the result of $\ltimes_\sim$ on {id}. Moreover, E_4 suggests that we should not allow projection on {item} altogether, except of course to allow for a semijoin $\ltimes_\sim$ to be applied. Finally, we should disallow projection on the emptyset, since this has no direct analogue in ASL. We thus arrive at the following fragment of SA, which we denote by $\mathsf{SA}^{\mathrm{search}}$.

Definition 27. The fragment $\mathsf{SA}^{\mathrm{search}}$ of SA is defined by the following rules:

—The operators $\ltimes_{=,\sim}$, $\pi_{\{\mathsf{item}\}}$ and $\pi_\emptyset$ are disallowed.
—The rule for $\ltimes_\sim$ is changed as follows:

$$\frac{E_1 : \{\mathsf{id}, \mathsf{item}\} \qquad E_2 : \{\mathsf{id}, \mathsf{item}\} \qquad \sim \in \mathcal{S}}{\pi_{\{\mathsf{id}\}}(E_1 \ltimes_\sim \pi_{\{\mathsf{item}\}} E_2) : \{\mathsf{id}\}}.$$

With these restrictions, we are ready to establish the connection between associative search and the semijoin algebra.

THEOREM 28. *Let $K \subseteq \mathcal{K}$ and $S \subseteq \mathcal{S}$. A search query is definable in* $\mathsf{ASL}(K^*, S)$ *if and only if it is definable in* $\mathsf{SA}^{\mathrm{search}}(K, S)$.

PROOF. The only-if direction follows from the observation that the translation from ASL to SA given in the proof of Proposition 24 stays within the fragment $\mathsf{SA}^{\mathrm{search}}$.

The if-direction is also proven by a translation, but a complication here is the translation of subexpressions with output schema {id, item}, since such intermediate results are not directly representable by the result of a search query (which can only return id's).

Formally, for each $\mathsf{SA}^{\mathrm{search}}$ expression E we construct a finite set χ_E of pairs (e, k), with e an $\mathsf{ASL}(K^*, S)$ expression and $k \in K^*$, such that for all D:

—if the output schema of E is {id, item}, then $E(rep(D))$ equals:

$$\bigcup_{(e,k)\in\chi_E} \{(o, i) \mid o \in e(D),\ i \in o \cap k\};$$

—if the output schema of E is {id}, then $E(rep(D))$ equals:

$$\bigcup_{(e,k)\in\chi_E} \{o \in e(D) \mid o \models k\}.$$

Of course the global SA expression E expressing the search query q has output schema Σ with $\{\mathsf{id}\} \subseteq \Sigma \subseteq \{\mathsf{id}, \mathsf{item}\}$, and thus q is defined in $\mathsf{ASL}(K^*, S)$ by:

$$(e_1 \text{ and } k_1) \text{ or } \ldots \text{ or } (e_n \text{ and } k_n),$$

where $\chi_E = \{(e_1, k_1), \ldots, (e_n, k_n)\}$.

We construct χ_E by induction as follows.

$$\begin{aligned}
\chi_T &:= \{(\star, \star)\} \\
\chi_{\sigma_l(E)} &:= \{(e, k \wedge l) \mid (e, k) \in \chi_E\} \\
\chi_{\pi_{\mathsf{id}}(E)} &:= \{(e \text{ and } k, \star) \mid (e, k) \in \chi_E\} \\
\chi_{E_1 \cup E_2} &:= \chi_{E_1} \cup \chi_{E_2} \\
\chi_{\pi_{\mathsf{id}}(E_1 \ltimes_{\sim} \pi_{\mathsf{item}} E_2)} &:= \{(e_1 \text{ and } \mathsf{link}\langle k_1 \sim k_2 \rangle\, e_2, \star) \mid (e_1, k_1) \in \chi_{E_1},\ (e_2, k_2) \in \chi_{E_2}\} \\
\chi_{E_1 \ltimes E_2} &:= \{(e_1 \text{ and } e_2 \text{ and } k_2, k_1) \mid (e_1, k_1) \in \chi_{E_1},\ (e_2, k_2) \in \chi_{E_2}\}.
\end{aligned}$$

For $E = E_1 - E_2$ we first need some terminology and notation. Let $\chi_{E_2} = \{(e_1, k_1), \ldots, (e_n, k_n)\}$. A *splitter* of E_2 is a pair (S^+, S^-) of disjoints subsets of $\{1, \ldots, n\}$ such that $S^+ \cup S^- = \{1, \ldots, n\}$. Then,

$$\begin{aligned}
\chi_{E_1 - E_2} := \{(e \text{ except } (e_{i_1} \text{ or } \ldots \text{ or } e_{i_r}), k \wedge \neg(k_{j_1} \vee \cdots \vee k_{j_s})) \mid (e, k) \in \chi_{E_1}, \\
(\{i_1, \ldots, i_r\}, \{j_1, \ldots, j_s\}) \text{ a splitter of } E_2\}.
\end{aligned}$$

The correctness of χ_E is proven by induction on E. We only show the reasoning for $E = E_1 - E_2$; the other cases are similar. Suppose that E_1 and E_2 both have schema {id, item}. Then, by induction hypothesis,

$$\begin{aligned}
&(o, i) \in E(rep(D)) \\
&\quad \Leftrightarrow (o, i) \in E_1(rep(D)) \text{ and } (o, i) \notin E_2(rep(D)) \\
&\quad \Leftrightarrow \text{there exists } (e, k) \in \chi_{E_1} : o \in e(D), i \in o \cap k \\
&\qquad \text{and for all } 1 \leq m \leq n : o \notin e_m(D) \text{ or } i \notin o \cap k_m \\
&\qquad \text{where } \chi_{E_2} = \{(e_1, k_1), \ldots (e_n, k_n)\} \\
&\quad \Leftrightarrow \text{there exists } (e, k) \in \chi_{E_1} : o \in e(D), i \in o \cap k \\
&\qquad \text{and a splitter } (\{i_1, \ldots, i_r\}, \{j_1, \ldots, j_s\}) \text{ for } E_2 \\
&\qquad \text{such that } o \notin e_m(D) \text{ for } m \in \{i_1, \ldots, i_r\} \\
&\qquad \text{and } i \notin o \cap k_m \text{ for } m \in \{j_1, \ldots, j_s\} \\
&\quad \Leftrightarrow (o, i) \in \bigcup_{(e,k) \in \chi_{E_1 - E_2}} \{(o', i') \mid o' \in e(D), i' \in o' \cap k\}.
\end{aligned}$$

The reasoning is similar when E_1 and E_2 both have schema {id}. □

We conclude this section with a remark concerning conjunctions in join conditions. We have defined simlinks as link conditions involving just a single condition $k \sim l$. But link conditions involving a conjunction of two or more such conditions are also very natural. For example, the query $\mathsf{link}\langle (k_1 \sim_1 l_1) \wedge (k_2 \sim_2 l_2) \rangle(e)$, applied to a dataspace D, would return those objects $o \in D$ for which there exists an object o' in $e(D)$ such that there exists a k_1-item $i \in o$ and an l_1-item $j \in o'$ such that $i \sim_1 j$, and there also exists a k_2-item $i \in o$ (not necessarily the same i) and an l_2-item $j \in o'$ (not necessarily the same j) such that $i \sim_2 j$.

Such conjunctive join conditions, however, can bring us outside the semijoin algebra. We can see this using the following example, which already gives a taste of what will be served in the next two sections.

Consider a chain of stores selling wine and cheese, with some stores located in France and others in Italy. We can model each store as a set of items. One item simply indicates the country France or Italy; other items are of the form (wine, v) listing all kinds of wine v sold by the store; and the remaining items are of the form (cheese, x) listing all kinds of cheese x sold by the store. Define the simrel samewine as the set of all identical pairs of the form ((wine, v), (wine, v)) for arbitrary wines v; define analogously the simrel samecheese as the set of all identical pairs of the form ((cheese, x), (cheese, x)) for arbitrary cheeses x.

Now consider the query "list all stores in France for which there exists a store in Italy selling some same wine and some same cheese", formally expressed as follows:

$$\text{France and link}\langle(\star\ \text{samewine}\ \star) \wedge (\star\ \text{samecheese}\ \star)\rangle(\text{Italy}).$$

This query is not expressible in the semijoin algebra. This can be proven formally as follows. Consider relational databases over two unary relations France and Italy, and two binary relations Wine and Cheese. The relation France (Italy) holds the ids of stores located in France (Italy), and the relation Wine (Cheese) holds the pairs (o, u) such that wine (cheese) u is sold in store o. Consider the two specific database instances D_1 and D_2, with the following content for D_1:

France
1

Italy
2

Wine	
1	bordeaux
2	bordeaux

Cheese	
1	roquefort
2	roquefort

So, in D_1, store 1 is French and store 2 is Italian and they offer exactly the same products (one wine and one cheese). Obviously then, the query applied to D_1 will return store 1. The content for D_2 is the following:

France
3

Italy
4
5
6

Wine	
3	bordeaux
4	bordeaux
5	sangiovese
6	sangiovese

Cheese	
3	parmesan
4	roquefort
5	parmesan
6	roquefort

Observe that our query has the empty answer on D_2. It can be verified, however, that $(D_1, 1)$ and $(D_2, 3)$ are *guarded bisimilar* and hence cannot be distinguished in the semijoin algebra [Leinders et al. 2005; Leinders and Van den Bussche 2007]. Consequently, any semijoin algebra query that returns 1 on D_1 will return 3 on D_2. Since our query does not have this property, it is not expressible in the semijoin algebra.

It seems very possible to define a fragment of the relational algebra appropriate for a generalization of Theorem 28 to conjunctions in link conditions. We leave this involved exercise to the interested reader.

3. APPLICATION 1: SEARCHING CLASSICAL RELATIONS

The theory presented in Section 2 holds for any concrete interpretation of $\mathcal{I}$. As an illustration of how this general theory can be applied, let us fix an infinite domain $\mathcal{V}$ of values and a relation schema Σ, that is, a finite set of attributes. We can use the set of attribute-value pairs $\Sigma \times \mathcal{V}$ as our set $\mathcal{I}$ of items. Note that a tuple $t : \Sigma \to \mathcal{V}$ is then an object over $\mathcal{I}$, and a finite relation R over

Σ is a dataspace over $\mathcal{I}$. Also note that conversely, not all objects over $\mathcal{I}$ are proper tuples, because in a proper tuple every attribute occurs exactly once. Hence, not all dataspaces in this setting are proper relations. Let us denote the subclass of dataspaces over $\Sigma \times V$ that are finite relations over Σ by $\mathcal{R}$.

3.1 Boolean Search

To search $\mathcal{R}$ we consider here the set of keywords:

$$\mathsf{LIT} := \{\star\} \cup \{\{(a, v)\} \mid a \in \Sigma, v \in \mathcal{V}\},$$

consisting of the wildcard $\star$ and the literal keywords over $\mathcal{I} = \Sigma \times \mathcal{V}$. Note that a literal keyword (a, v) corresponds to the relational selection operator $\sigma_{a=v}$. Indeed, for all $R \in \mathcal{R}$ Definition 4 specializes to:

$$(a, v)(R) = \{t \in R \mid t(a) = v\},$$

since R consists only of tuples over Σ. So, the language $\mathsf{BSL}(\mathsf{LIT})$ evaluated over relations in $\mathcal{R}$ corresponds to the fragment of the relational algebra consisting of just the three operators of constant selection, union, and difference.

It turns out that our notion of K-distinguishing search queries corresponds in this setting to the so-called fully C-generic queries. Fully generic queries without constants have been introduced and investigated (in a much richer setting than mere selection queries) by Beeri, Milo and Ta-Shma [Beeri et al. 1996; Beeri et al. 1997]; here we consider the version with constants.

Definition 29 (*C-epimorphism*). Let $C \subset \mathcal{V}$ be a finite set of constants. A *C-epimorphism* is a mapping $f : \mathcal{V} \to \mathcal{V}$ such that both $f|_C$ and $f^{-1}|_C$ are the identity on C. We extend $f : \mathcal{V} \to \mathcal{V}$ to items, objects, and dataspaces in the canonical, pointwise manner:

$$\begin{aligned} f(a, v) &:= (a, f(v)), \\ f(o) &:= \{f(a, v) \mid (a, v) \in o\}, \\ f(R) &:= \{f(o) \mid o \in R\}. \end{aligned}$$

Now a search query q is said to be *fully C-generic* (on the class $\mathcal{R}$) if for any relation R over Σ and any C-epimorphism f, we have $q(f(R)) = f(q(R))$.

Note that f is a surjective homomorphism from R to $f(R)$, which justifies our use of the term, epimorphism, which traditionally means surjective homomorphism.

We establish:

PROPOSITION 30. *Let C be a finite set of constants, and let $K = \{\{(a, v)\} \mid a \in \Sigma, v \in C\}$. The following are equivalent for any search query q:*

(1) *q is K-distinguishing on the class $\mathcal{R}$;*
(2) *q is additive and fully C-generic on $\mathcal{R}$.*

PROOF. $(1 \to 2)$ Since q is K-distinguishing on $\mathcal{R}$, we know that for every $R \in \mathcal{R}$ and every $t \in R$ we have $t \in q(R)$ if and only if $t \in q(\{t\})$. This readily implies additivity. It remains to show that q is fully C-generic on $\mathcal{R}$. Theretoward, fix

$R \in \mathcal{R}$ arbitrarily and let f be a C-epimorphism. Observe that $t \simeq_K f(t)$ for all $t \in R$. Indeed, let $k = \{(a, v)\}$ be an arbitrary keyword in K with $a \in \Sigma$ and $v \in C$. If $t \models k$, then $t(a) = v$. Since $f|_C$ is the identity on C, $f(v) = v$ and hence $f(t)(a) = v$. Thus $f(t) \models k$. Conversely, if $f(t) \models k$ then $f(t)(a) = v$. Since $f^{-1}|_C$ is the identity on C, also $t(a) = v$. Therefore, $t \models k$. As such, t satisfies the same keywords in K as $f(t)$. Since q is K-distinguishing, we hence have $t \in q(R) \Leftrightarrow f(t) \in q(f(R))$, which implies $q(f(R)) = f(q(R))$, as desired.

$(2 \rightarrow 1)$ Let R and R' be arbitrary relations in $\mathcal{R}$ and let $t \in R$ and $t' \in R'$ such that $t \simeq_K t'$. We need to show that $t \in q(R) \Leftrightarrow t' \in q(R')$. Theretoward, first observe that for all $a \in \Sigma$, if $t(a) \in C$ then $t'(a) = t(a)$ (since $t \simeq_K t'$) and conversely if $t'(a) \in C$ then $t(a) = t'(a)$ (for the same reason). Then construct the tuple u such that for all $a \in \Sigma$:

— if $t(a) \in C$ then $u(a) = t(a) = t'(a)$; and
— if $t(a) \notin C$ then $u(a) \notin C$ and $u(a) \neq u(b)$ for every $b \in \Sigma$ distinct from a.

It is readily verified that there exists C-epimorphism f and f' such that $f(u) = t$ and $f'(u) = t'$. Now reason as follows:

$$\begin{aligned} t \in q(R) &\Leftrightarrow t \in q(\{t\}) \\ &\Leftrightarrow f(u) \in q(\{f(u)\}) \\ &\Leftrightarrow u \in q(\{u\}) \\ &\Leftrightarrow f'(u) \in q(\{f'(u)\}) \\ &\Leftrightarrow t' \in q(\{t'\}) \\ &\Leftrightarrow t' \in q(R'). \end{aligned}$$

Hence the proposition. □

From Proposition 30; Corollary 12; and the fact that BSL(LIT) corresponds to the fragment of the relational algebra consisting of just the three operators of constant selection, union, and difference, we then obtain:

COROLLARY 31. *A search query q on relations over Σ is definable in the relational algebra using only the operators constant selection, union, and difference, if and only if q is additive and fully C-generic for some finite set C of constants.*

The main purpose of this modest theorem is to illustrate that our general theory can be effectively applied and connected to earlier work. Moving towards associative search over classical relations, current systems implement languages approaching the full relational algebra (e.g., Golenberg et al. [2008] and Qin et al. [2009]), the expressive power of which is already well understood [Abiteboul et al. 1995].

4. APPLICATION 2: SEARCHING ATTRIBUTE-VALUE DATASPACES

We next apply the abstract theory of Section 2 to the concrete setting of attribute-value dataspaces as they have been investigated in the literature [Halevy et al. 2006; Dong and Halevy 2007; Dittrich and Vaz Salles 2006]. In this setting, items are attribute-value pairs as in the relational setting of Section 3, so $\mathcal{I} = \Sigma \times \mathcal{V}$. The important difference, however, is that the universe

of attributes Σ can now be infinite; there is no longer a fixed finite relation schema. Of course each dataspace is finite, so in each dataspace just a finite number of attributes will occur, but these attributes can vary from dataspace to dataspace and even from object to object. Moreover, attributes can appear multiple times in the same object, with different values. So, an object really is just a nonempty finite set of items, without any restriction.

The dataspace models in the literature [Halevy et al. 2006; Dong and Halevy 2007; Dittrich and Vaz Salles 2006] give each object an id, and naturally represent a dataspace D as a set of triples $\{(o, a, v) \mid o \in D \text{ and } (a, v) \in o\}$. We argue that our view of a dataspace as just a set of objects is equivalent. Indeed, we just showed how to go from our representation to the set of triples; if one wants to go in the converse direction, the only difficulty one may encounter is that there might be two object ids in the triple set with exactly the same set of associated (a, v) pairs. In that case we can add an explicit id attribute to the objects, so that the sets become distinct. Having explicit id attributes is also necessary when we need to represent links between objects based on ids. We will see an example of such linking later (Example 38).

4.1 Boolean Search

The next question is what keywords to use for searching attribute-value dataspaces. Surely we need all literal keywords (a, v) (as in the classical relational case), so that we can formulate basic queries like "retrieve all persons who live in Belgium, like the beer Duvel, but do not like the beer Heineken":[1]

$$((\mathsf{country}\colon \mathsf{Belgium})\ \mathsf{and}\ (\mathsf{likes}\colon \mathsf{Duvel}))\ \mathsf{except}\ (\mathsf{likes}\colon \mathsf{Heineken}).$$

We also want negation separately on attributes and values: for example,

$$(\mathsf{likes}\colon \neg\mathsf{Heineken})$$

retrieves objects containing a value for attribute likes that is different from Heineken, and $(\neg\mathsf{likes}\colon \mathsf{Heineken})$ retrieves objects containing Heineken as the value of an attribute different from likes. Note that, since the set of attributes is not fixed, we cannot express the last example by a disjunction using all possible attributes other than likes. Similarly, we need wildcards on values as well as on attributes, so $(\star\colon \mathsf{Belgium})$ retrieves all objects with value Belgium for some attribute. Finally, we need disjunctions such as $(\mathsf{likes}\colon \neg(\mathsf{Heineken} \vee \mathsf{Budweiser}))$. (We only need negated disjunctions; positive disjunctions can already be expressed in BSL using or.)

To sum up, we propose the following system of keywords for attribute-value pairs.

[1]To improve readability, we will write attribute-value pairs (a, v) as $(a\colon v)$, conforming more to a programming language-like syntax.

Definition 32. First define the following abbreviations for $a \in \Sigma$, $v \in \mathcal{V}$, $A \subseteq \Sigma$, and $V \subseteq \mathcal{V}$:

$$\begin{aligned}(a\colon v) &:= \{(a, v)\}\\ (a\colon \neg V) &:= \{(a, v) \mid v \in \mathcal{V} - V\}\\ (\neg A\colon v) &:= \{(a, v) \mid a \in \Sigma - A\}\\ (\neg A\colon \neg V) &:= \{(a, v) \mid a \in \Sigma - A, v \in \mathcal{V} - V\}.\end{aligned}$$

Note that the wildcard $\star$ is readily obtained by $(\neg\emptyset, \neg\emptyset)$. The set AV of *attribute–value keywords* is the set consisting of all keywords of the form:

$$(a\colon v) \mid (a\colon \neg V) \mid (\neg A\colon v) \mid (\neg A\colon \neg V),$$

where $a \in \Sigma$, $v \in \mathcal{V}$, and $A \subseteq \Sigma$, $V \subseteq \mathcal{V}$ are finite.

A first indication of the flexibility of this keyword system is that we do not need to add Boolean combinations:

PROPOSITION 33. $\mathsf{BSL}(\mathsf{AV})$ *is equivalent to* $\mathsf{BSL}(\mathsf{AV}^*)$*, where* AV^* *denotes Boolean closure of* AV *keywords (cf. Definition 25).*

PROOF. It is readily verified that every Boolean combination of AV keywords amounts to a disjunction of AV keywords. Such a disjunction can be expressed in BSL using or. To illustrate using a few concrete examples:

$$\begin{aligned}\neg(a\colon v) &\equiv (\neg a\colon \neg\emptyset)\ \mathsf{or}\ (\neg\emptyset\colon \neg\{v\})\\ \neg(\neg A\colon v) &\equiv (A\colon \neg\emptyset)\ \mathsf{or}\ (\neg\emptyset\colon \neg\{v\})\\ (\neg A\colon v) \vee (a\colon \neg V) &\equiv (\neg A\colon v)\ \mathsf{or}\ (a\colon \neg V)\\ \neg\big((\neg A\colon v) \vee (a\colon \neg V)\big) &\equiv (A \setminus \{a\}\colon \neg\emptyset)\ \mathsf{or}\ (A \cap \{a\}\colon V)\ \mathsf{or}\ (\neg\{a\}\colon \neg\{v\})\ \mathsf{or}\ (a\colon V).\end{aligned}$$ □

Since we have the wildcard, Corollary 9 applies, so we know that $\mathsf{BSL}(\mathsf{AV})$ defines exactly all search queries that are K-distinguishing for some finite set $K \subseteq \mathsf{AV}$. Recall that a search query q is K-distinguishing if it is invariant under the equivalence $\simeq_K$ on objects (Definition 7). In the attribute-value setting, we can formulate a more intuitive alternative to this equivalence relation, directly in terms of attributes and values, rather than AV keywords. The idea, similar to full genericity, is that only a finite set of attributes and values can be distinguished.

Definition 34. Let W be a finite set of attributes and values. Let $\diamond$ be a blank value, which is an arbitrary element not in W. For an attribute or value x, define

$$blank_W(x) := \begin{cases} x & \text{if } x \in W\\ \diamond & \text{otherwise.}\end{cases}$$

We extend $blank_W$ to attribute-value pairs, objects, and dataspaces in the canonical, pointwise manner:

$$\begin{aligned} blank_W(a, v) &:= (blank_W(a), blank_W(v)), \\ blank_W(o) &:= \{blank_W(a, v) \mid (a, v) \in o\}, \\ blank_W(D) &:= \{blank_W(o) \mid o \in D\}. \end{aligned}$$

Two objects o_1 and o_2 are called *W-equivalent* if $blank_W(o_1) = blank_W(o_2)$. We denote this by $o_1 \simeq_W o_2$. We now say that a search query q is *W-distinguishing* if for any two dataspaces D_1 and D_2 and objects $o_1 \in D_1$ and $o_2 \in D_2$ that are W-equivalent, we have $o_1 \in q(D_1)$ iff $o_2 \in q(D_2)$.

Example 35. Intuitively, $blank_W$ replaces all attributes and values not in W by the same constant $\diamond$. Two objects are W-equivalent if they are the same after blanking. To illustrate, reconsider the attribute-value dataspace of Figure 1. Then $blank_{\{\mathsf{Bill}\}}(o_1)$ and $blank_{\{\mathsf{Bill}\}}(o_3)$ both yield:

◇:	Bill
◇:	◇

Hence, $o_1 \simeq_{\{\mathsf{Bill}\}} o_3$. In contrast, $blank_{\{\mathsf{Bill,author}\}}(o_1)$ and $blank_{\{\mathsf{Bill,author}\}}(o_3)$ yield respectively:

◇:	Bill
◇:	◇

author:	Bill
author:	◇
◇:	◇

Hence $o_1 \not\simeq_{\{\mathsf{Bill,author}\}} o_3$.

PROPOSITION 36. *A search query is K-distinguishing for a finite set $K \subseteq \mathsf{AV}$ if and only if, it is W-distinguishing for a finite set of attributes and values W.*

PROOF. The crux of the matter is that if two objects are K-equivalent, for some finite set of AV keywords K, then they are W-equivalent, where W is the finite set of attributes and values explicitly mentioned in the keywords in K. Conversely, if two objects are W-equivalent, for some finite set of attributes and values W, then they are K-equivalent, where K is the finite set of all AV keywords that can be constructed using the elements in W. □

4.2 Associative Search

Let us now turn to associative search in the attribute-value context. What are the simrels needed to join objects in an attribute-value dataspace? Focusing on equijoins, there are three natural possibilities: two pairs can be compared on their values, on their attributes, or on both together. So, for the set $\mathcal{S}$ of simrels in the attribute-value setting, we will use the set $\{\mathsf{eq}, \mathsf{eq\text{-}attr}, \mathsf{eq\text{-}val}\}$, defined as follows:

Definition 37. An *eqrel* (short for equality relation) is one of the three following simrels on attribute–value pairs:

$$(a, v)\ \mathsf{eq}\ (b, w) \Leftrightarrow a = b \text{ and } v = w,$$
$$(a, v)\ \mathsf{eq\text{-}attr}\ (b, w) \Leftrightarrow a = b,$$
$$(a, v)\ \mathsf{eq\text{-}val}\ (b, w) \Leftrightarrow v = w.$$

We denote by EQ the set {eq, eq-attr, eq-val} of all eqrels.

Example 38. For example, the following is an expression in the associative search language ASL(AV, EQ):

$$\mathsf{link}\langle(\mathsf{name}:\star)\ \mathsf{eq\text{-}val}\ (\mathsf{author}:\star)\rangle(\mathsf{published\ in}:\mathsf{ICDT\ 2009})$$

It defines the query that retrieves all authors of objects published in ICDT 2009. More precisely, it retrieves all objects with a value for attribute name that equals a value for attribute author in an object containing the item (published in, ICDT 2009).

It should be noted that in other dataspace models based on attribute-value pairs [Dong and Halevy 2007; Dittrich and Vaz Salles 2006], objects are not joined using eqrel simlinks, but through explicit named links (edges) between objects. So there, a dataspace is not merely a set of objects, but a *directed graph* of objects.

Example 39. As discussed in the opening of Section 4, objects in the dataspace model of Dong and Halevy are represented by explicit identifiers, and relationships between objects are represented as triples [Dong and Halevy 2007]. For example, the subset $\{o_1, o_4\}$ of the attribute-value dataspace given in Figure 1 is captured in the Dong-Halevy model as:

$$\begin{array}{llll} \{\langle \text{o1}, & \text{name}, & \text{Bill}\rangle, \\ \langle \text{o1}, & \text{email}, & \text{bill@gmail}\rangle, \\ \langle \text{o1}, & \text{email}, & \text{bill@yahoo}\rangle, \\ \langle \text{o1}, & \text{likes}, & \text{o4}\rangle, \\ \langle \text{o4}, & \text{drink}, & \text{vodka}\rangle, \\ \langle \text{o4}, & \text{origin}, & \text{Russia}\rangle\}. \end{array}$$

Note in particular the edge between object o_1 and o_4 defined by ⟨o1,likes, o4⟩.

Using eqrel simlinks as we do, an explicit graph model is redundant. Indeed, as illustrated in Example 38, named edges (for instance linking papers to their authors) can easily be represented using id attributes for objects and "pointer" attributes (name and author) having these ids as values.

We next show that the abstract bisimulation, Lemma 19, can well be applied in the present setting as an aid to understand the limits of expressive power of ASL(AV, EQ). For example, consider the generic equality selection query $q_{a=b}$, for two attributes a and b, defined as follows:

$$q_{a=b}(D) = \{o \in D \mid \exists v : (a, v) \in o \text{ and } (b, v) \in o\}.$$

It is not immediately clear whether or not this query is definable in ASL(AV, EQ); it turns out it is not.

PROPOSITION 40. *$q_{a=b}$ is not definable in* ASL(AV, EQ).

PROOF. In a nutshell, the proof consist of showing that $q_{a=b}$ is not bisimulation invariant relative to any finite set of AV keywords and eqrel simlinks, and then invoking Lemma 19. Since our aim is to show that $q_{a=b}$ is not definable in ASL using any set of AV keywords and eqrel simlinks, the proof technique is necessarily more abstract than that followed in the proof of Proposition 26, where it sufficed to restrict ourselves to carefully constructed sets of keywords and simlinks, respectively.

Concretely, let e be an arbitrary expression in ASL(AV, EQ), let K be the finite set of keywords mentioned in e (be it as an expression or as a link condition), let:

$$L = \{k \sim l \mid k, l \in K, \sim \in \mathsf{EQ}\},$$

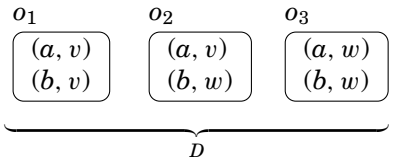
and let $n = \mathit{depth}(e)$ be the nesting depth of link expressions in e. To see that e cannot define $q_{a=b}$, first define, for every $M \subseteq K$, the set $[\![M]\!]$ of pairs matched by all keywords in M and none of the keywords in $K - M$:

$$[\![M]\!] := (\Sigma \times \mathcal{V}) \cap \bigcap_{k \in M} k - \bigcup_{k' \in K - M} k'.$$

Then the set $\{[\![M]\!] \mid M \subseteq K\}$ forms a partition of $\Sigma \times \mathcal{V}$. In particular, $\Sigma \times \mathcal{V} = \bigcup_{M \subseteq K} [\![M]\!]$. Since the subset $\{(a, u) \mid u \in \mathcal{V}\}$ of $\Sigma \times \mathcal{V}$ is infinite and since there are only a finite number of subsets of K, there must be at least one $M^a \subseteq K$ for which $[\![M^a]\!] \cap \{(a, u) \mid u \in \mathcal{V}\}$ is infinite. Then let $\{M_1^b, \ldots, M_m^b\} = \{M \subseteq K \mid \exists u \in \mathcal{V} : (b, u) \in [\![M]\!]\}$. Since $\{u \mid (a, u) \in [\![M^a]\!]\}$ is infinite, since $\mathcal{V} = \bigcup_{m=1}^{n} \{u \mid (b, u) \in [\![M_i^b]\!]\}$, and since m is finite, there must exist some $1 \leq i \leq m$ such that:

$$\{u \mid (a, u) \in [\![M^a]\!]\} \cap \{u \mid (b, u) \in [\![M_i^b]\!]\},$$

is infinite. Then fix v, w in this intersection with $v \neq w$. By construction, $(a, v), (a, w) \in [\![M^a]\!]$ and $(b, v), (b, w) \in [\![M_i^b]\!]$, that is, (a, v) and (a, w) satisfy the same keywords over K and similarly for (b, v) and (b, w). Now fix D as follows.



To prove the proposition, it suffices to show $(D, o_1) \rightleftarrows_n^{K,L} (D, o_2)$. Indeed, then $o_1 \in e(D) \Leftrightarrow o_2 \in e(D)$ by Lemma 19, while $o_1 \in q_{a=b}(D)$ but $o_2 \notin q_{a=b}(D)$. As such, e cannot define $q_{a=b}$.

We actually show that $(D, o_1) \rightleftarrows_n^{K,L} (D, o_2)$ and $(D, o_1) \rightleftarrows_n^{K,L} (D, o_3)$ by simultaneous induction on n. When $n = 0$, $o_1 \simeq_K o_2$ and $o_1 \simeq_K o_3$ by construction ((a, v) matches the same keywords as (a, w) and (b, v) matches the same keywords as (b, w)), and hence $(D, o_1) \rightleftarrows_0^{K,L} (D, o_2)$ and $(D, o_1) \rightleftarrows_0^{K,L} (D, o_3)$. Inductively, it suffices to check the forth and back properties.

(*Forth*) Suppose that there is some $\lambda \in L$ and some $p_1 \in D$ such that $\lambda(o_1, p_1)$. We show that there exists $p_2, p_3 \in D$ such that (1) $\lambda(o_2, p_2)$ and $(D, p_1) \rightleftarrows^{K,L}_{n-1} (D, p_2)$ and (2) $\lambda(o_3, p_3)$ and $(D, p_1) \rightleftarrows^{K,L}_{n-1} (D, p_3)$.

— Case $\lambda = k$ eq l. Then $o_1 \cap k \cap p_1 \cap l$ is nonempty. There are two possibilities. (1) When (a, v) is in the intersection, we can take $p_2 := p_1$ and $p_3 := o_3$. Indeed, to see that it suffices to take $p_2 := p_1$, observe that (a, v) is also in $o_2 \cap k \cap p_2 \cap l$ (which is hence nonempty). Moreover, since $\rightleftarrows$ is an equivalence relation and $p_1 = p_2$, also $(D, p_1) \rightleftarrows^{K,L}_{n-1} (D, p_2)$. To see that it suffices to take $p_3 := o_3$, observe that by construction also $(a, w) \in o_3 \cap k \cap p_3 \cap l$ since (a, v) and (a, w) satisfy the same keywords in K. Moreover, since p_1 is either o_1,o_2, or o_3; since $(D, o_1) \rightleftarrows^{K,L}_{n-1} (D, o_2)$ and $(D, o_1) \rightleftarrows^{K,L}_{n-1} (D, o_3)$ (by induction hypothesis); since also $(D, o_2) \rightleftarrows^{K,L}_{n-1} (D, o_3)$ (by transitivity of $\rightleftarrows$); and since $(D, o_3) \rightleftarrows^{K,L}_{n-1} (D, o_3)$ (by reflexivity of $\rightleftarrows$), certainly $(D, p_1) \rightleftarrows^{K,L}_{n-1} (D, p_3)$. (2) When (b, v) is in the intersection, a symmetric argument shows that it suffices to take $p_2 := o_2$ and $p_3 := p_1$.

— Case $\lambda = k$ eq-attr l. Then $\pi_1(o_1 \cap k) \cap \pi_1(p_1 \cap l)$ is nonempty. As such, either a is in the intersection, or b is. In both events, it can be seen using a similar argument as in the case $\lambda = k$ eq l, that it suffices to take $p_2 := p_1$ and $p_3 := p_1$.

— Case $\lambda = k$ eq-val l. Then $\pi_2(o_1 \cap k) \cap \pi_2(p_1 \cap l)$ is nonempty. Since v is the only word in $\pi_2(o_1)$, $\pi_2(o_1 \cap k) \cap \pi_2(p_1 \cap l) = \{v\}$. There are four possibilities why this is so:
(a) $(a, v) \in o_1 \cap k$ and $(a, v) \in p_1 \cap l$;
(b) $(a, v) \in o_1 \cap k$ and $(b, v) \in p_1 \cap l$;
(c) $(b, v) \in o_1 \cap k$ and $(a, v) \in p_1 \cap l$;
(d) $(b, v) \in o_1 \cap k$ and $(b, v) \in p_1 \cap l$.
Similar arguments as in the case $\lambda = k$ eq l, show that it suffices to take $p_2 := o_2$ and $p_3 := o_3$ in the first event; $p_2 := o_1$ and $p_3 := o_2$ in the second; $p_2 := o_3$ and $p_3 := o_3$ in the third; and $p_2 := o_2$ and $p_3 := o_3$ in the fourth.

(*Back*) Similar to Forth. □

4.3 On the Correspondence with the Semijoin Algebra

We conclude this section by returning to the equivalence between ASL and the semijoin algebra. Recall that in Section 2.3 we defined the semijoin algebra to work on abstract dataspaces represented as binary relations over the schema {id, item}. While the equivalence of ASL and $\mathsf{SA}^{\mathsf{search}}$ (Theorem 28) can be directly applied to the attribute-value setting, it is not so natural to store attribute-value pairs in a single item column. It is more natural to represent attribute-value dataspaces as sets of triples, that is, as ternary relations over the schema {id, attr, val}. Also the RDF query language SPARQL works over such ternary relations [W3C 2008; Gutierrez et al. 2004; Pérez et al. 2009]; RDF graphs can also be viewed as a dataspace model [Dong and Halevy 2007].

So, it is worthwhile to define an alternative to $\mathsf{SA}^{\mathsf{search}}$ working on ternary relations. An added simplification is that, since we are working with eqrel

simlinks rather than simlinks based on general abstract simrels, we will no longer need the $\sim$-semijoin operator and will have enough with the standard natural semijoin.

Definition 41. The fragment $\mathsf{SA}^{\mathrm{AV}}$ of the semijoin algebra, defined on relations T : {id, attr, val}, is defined by the following grammar:

$$E ::= T \mid \sigma_{\mathsf{attr}=c}(E) \mid \sigma_{\mathsf{val}=c}(E) \mid E \cup E \mid E - E \mid \\ \mid \pi_\alpha(E) \mid E \ltimes \pi_{\mathsf{id}}(E) \mid \pi_{\{\mathsf{id}\}}(E \ltimes \pi_\beta(E)),$$

where c ranges over attribute and value constants; α is either {id}, {id, attr}, or {id, val}; and β is either {attr}, {val}, or {attr, val}.

The semantics of $\ltimes$ is the standard natural semijoin on equality of common attributes.

We have the following analog of Theorem 28.

THEOREM 42. *A search query is definable in* ASL(AV, EQ)*, if and only if it is definable in* $\mathsf{SA}^{\mathrm{AV}}$*, where we regard a dataspace* D *as a ternary relation* $\{(o, a, v) \mid o \in D \text{ and } (a, v) \in o\}$.

PROOF. The only-if direction is similar to the proof of Theorem 28. AV keywords are expressed using combinations of constant selections using union and difference. Simlinks based on eq-attr, eq-val, or eq are expressed using semijoin with projection on the right (the set β in the syntax definition) equal to {attr}, {val}, or {attr, val}, respectively.

The if-direction is also similar, but we have the added complication that Boolean combinations of keywords are not directly available in the AV setting. Yet, because of Proposition 33, we can simulate them in the language. Proposition 33 is only formulated for BSL(AV), but the same argument holds for ASL(AV, EQ), because link distributes over a disjunction of keywords used in a simlink:

$$\mathsf{link}\langle(\varphi_1 \vee \varphi_2) \sim \psi\rangle(e) = \mathsf{link}\langle\varphi_1 \sim \psi\rangle(e) \text{ or } \mathsf{link}\langle\varphi_2 \sim \psi\rangle(e).$$

Semijoins are translated to simlinks using an eqrel depending on β: if $\beta = \{\mathsf{attr}\}$ we use eq-attr, if $\beta = \{\mathsf{val}\}$ we use eq-val, and if $\beta = \{\mathsf{attr}, \mathsf{val}\}$ we use eq. □

5. DISCUSSION

Our goal has been to provide the beginnings of a theoretical foundation for search and associative search queries, motivated by the ubiquity of such queries in everyday information systems. Our approach has been to investigate search queries as restricted kinds of database queries, and to use the tools and the concepts already developed in the theory of database queries.

We first presented a general abstract theory, then applied it to the concrete setting of attribute-value dataspaces. It would be interesting to conduct a similar application to the XML data model, for example, with XPath playing the role of relational algebra.

Mainly inspired by dataspaces [Dong and Halevy 2007], we have focused on selection queries, that is, queries that always return a subset of the original objects. Current approaches to keyword search on structured databases

[Golenberg et al. 2008; Qin et al. 2009; we give just two recent references] return tuples of objects that are related by patterns. In the semijoin algebra one can express such patterns as long as they are not cyclic, but the patterns themselves cannot be returned. It remains to be investigated if and how our theory should be extended so that patterns can be returned. Of course, one can simply move to the full relational algebra, but then there is less news to discover.

It would also be interesting to look at a similar theory of search queries on RDF graphs (ternary relations, as discussed in Section 4.3). Indeed, RDF graphs can be viewed as a particular instance of the classical relations, and hence our results of Section 3.1 for Boolean search already apply here. The expressivity of SPARQL, which is a kind of specialized relational algebra for RDF, has been studied recently [Fletcher 2008; Gutierrez et al. 2004; Pérez et al. 2009]. Given their close similarity to AV dataspaces, it might also be fruitful to investigate specializations of our semijoin algebra fragment $\mathsf{SA}^{\mathrm{AV}}$ into interesting fragments of SPARQL, or of its navigational extensions (e.g., Alkhateeb et al. [2009] and Pérez et al. [2008]).

ACKNOWLEDGMENTS

The authors thank the reviewers for their many constructive comments that helped improve the presentation of this article.

REFERENCES

ABITEBOUL, S., HULL, R., AND VIANU, V. 1995. *Foundations of Databases*. Addison-Wesley.

AGRAWAL, R., SOMANI, A., AND XU, Y. 2001. Storage and querying of e-commerce data. In *Proceedings of the 27th International Conference on Very Large Data Bases*. 149–158.

AHO, A., AND ULLMAN, J. 1979. Universality of data retrieval languages. In *Conference Record of the 6th ACM Symposium on Principles of Programming Languages*. 110–120.

ALKHATEEB, F., BAGET, J.-F., AND EUZENAT, J. 2009. Extending SPARQL with regular expression patterns (for querying RDF). *J. Web Sem. 7,* 2, 57–73.

BEERI, C., MILO, T., AND TA-SHMA, P. 1996. On genericity and parametricity. In *Proceedings of the 15th ACM Symposium on Principles of Database Systems*. 104–116.

BEERI, C., MILO, T., AND TA-SHMA, P. 1997. Towards a language for the fully generic queries. In *Database Programming Languages*, S. Cluet and R. Hull, Eds. Lecture Notes in Computer Science. Springer, 239–259.

BLACKBURN, P., DE RIJKE, M., AND VENEMA, Y. 2001. *Modal Logic*. Cambridge University Press.

BLACKBURN, P., VAN BENTHEM, J., AND WOLTER, F., Eds. 2007. *Handbook of Modal Logic*. Elsevier.

CHANDRA, A., AND HAREL, D. 1980. Computable queries for relational data bases. *J. Comput. Syst. Sci. 21,* 2, 156–178.

DECANDIA, G., HASTORUN, D. JAMPANI, M., KAKULAPATI, G., LAKSHMAN, A., PILAHIN, A., SIVASUBRAMANIAN, S., VOSSHALL, P., AND VOGELS, W. 2007. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev. 41,* 6, 205–220.

DITTRICH, J.-P., AND VAZ SALLES, M. 2006. iDM: A unified and versatile data model for personal dataspace management. In *Proceedings of the 32nd International Conference on Very Large Data Bases*. 367–378.

DONG, X., AND HALEVY, A. 2007. Indexing dataspaces. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 43–54.

FLETCHER, G. H. L. 2008. An algebra for basic graph patterns. *Logic in Databases* (LID). *Informal Proceedings.*

FLETCHER, G. H. L., VAN DEN BUSSCHE, J., VAN GUCHT, D., AND VANSUMMEREN, S. 2009. Towards a theory of search queries. In *Proceedings of the 12th International Conference on Database Theory*. 201–211.

GOLENBERG, K., KIMELFELD, B., AND SAGIV, Y. 2008. Keyword proximity search in complex data graphs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 927–940.

GORANKO, V., AND OTTO, M. 2007. Model theory of modal logic. *in Handbook of Hoda Logic,* P. Blackburn, J. van Benthem, and F. Wolter, Ed., Chapter 5.

GUTIERREZ, C., HURTADO, C., AND MENDELZON, A. 2004. Foundations of semantic Web databases. In *Proceedings of the 23rd ACM Symposium on Principles of Database Systems*. 95–106.

HALEVY, A., FRANKLIN, M., AND MAIER, D. 2006. Principles of dataspace systems. In *Proceedings of the 25th ACM Symposium on Principles of Database Systems*. 1–9.

JAGADISH, H., CHAPMAN A., ELKISS, A., JAYAPANDIAN, M., LI, Y., NANDI, A., AND YU, C. 2007. Making database systems usable. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 13–24.

JAIN, M., MENDHEKAR, A., AND VAN GUCHT, D. 1995. A uniform data model for relational data and meta-data query processing. In *Advances in Data Management '95*, Tata McGraw-Hill, 146–165.

LEINDERS, D., MARX, M., TYSZKIEWICZ, J., AND VAN DEN BUSSCHE, J. 2005. The semijoin algebra and the guarded fragment. *J. Logic Lang. Inform. 14*, 331–343.

LEINDERS, D., AND VAN DEN BUSSCHE, J. 2007. On the complexity of division and set joins in the relational algebra. *J. Comput. Syst. Sci. 73,* 4, 538–549.

LITWIN, W., KETABCHI, M., AND KRISHNAMURTHY, R. 1991. First order normal form for relational databases and multidatabases. *SIGMOD Rec. 20,* 4, 74–76.

MANNING, C., RAGHAVAN, P., AND SHÜTZE, H. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

OLSTON, C., REED, B., SRIVASTAVA, U., KUMAR, R., AND TOMKINS, A. 2008. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 1099–1110.

PÉREZ, J., ARENAS, M., AND GUTIERREZ, C. 2008. nSPARQL: A navigational language for RDF. In *Proceedings of the International Semantic Web Conference*. 66–81.

PÉREZ, J., ARENAS, M., AND GUTIERREZ, C. 2009. Semantics and complexity of SPARQL. *ACM Trans. Datab. Syst. 34,* 3.

QIN, L., YU, J.X., AND CHANG, L. 2009. Keyword search in databases: the power of RDBMS. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 681–694.

ROSS, K., AND JANEVSKI, A. 2005. Querying faceted databases. In *Proceedings of the 2nd International Workshop on Semantic Web and Databases*. Lecture Notes in Computer Science, vol. 3372, Springer, 199–218.

SARAWAGI, S., AND KIRPAL, A. 2004. Efficient set joins on similarity predicates. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 743–754.

ULLMAN, J. 1989. *Principles of Database and Knowledge-Base Systems*. Vol. II. Computer Science Press.

W3C. 2004. RDF primer. W3C Recommendation.

W3C. 2008. SPARQL query language for RDF. W3C Recommendation.

Received October 2009; revised March 2010; accepted April 2010