Towards a theory of search queries

George H.L. Fletcher Jan Van den Bussche Dirk Van Gucht
Stijn Vansummeren

Abstract

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 characterise 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. Specifically, we show that the basic search language extended with associative search can define exactly the queries definable in a restricted fragment of the semijoin algebra working on an explicit relational representation of the dataspace.

1 Introduction

In most current information systems, such as Web search engines, ecommerce 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. One can say that searching has become the norm. Searching is expressed by means of keywords which 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. Indeed, fully-fledged first-order logic is the norm (cf. Codd's relational algebra and calculus), and contemporary languages such as SQL/PSM or XQuery are even computationally complete.

Database queries are a major theme of database theory [3]. Database queries, in general, are generic mappings from databases to relations, where 'generic' refers to invariance under isomorphisms [4, 10]. Many classes of database queries have been identified, and have been 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 paper 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 [11, 12, 17] are a new type of databases and are characterized by a very loosely structured data model, geared towards the management of data coming from a diverse set of sources. Dataspaces are 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.

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 [23, Chapter 17]). Recently, Jagadish [18] has 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 paper, we try to understand formally 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 are *not* doing in this paper. We are not investigating here how searching can be implemented efficiently. Rather, we are focusing on expressive power. Also, because of this focus, we are ignoring here other important issues that have been investigated in research on keyword search in relational, tree-structured (XML), and semistructured (graph) databases [14, we give just one recent reference]. 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 mentioned above), and ranking the results of a keyword search.

Concretely, the contributions of this paper 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 [3]; we have tried here to get at the right genericity concept for search queries.

3. We apply the above 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 using as filters on such attribute–value pairs the classical selection conditions "attribute equals constant", we obtain as basic search queries precisely the class of search queries that are "fully generic" in the sense of Beeri, Milo and Ta-Shma [6, 7]. (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 amounts to adding a link operator to the basic search language, much in the spirit of modal logic [8, 9]. Associative search is indeed a feature of most dataspace and keyword search query languages proposed in the literature (see earlier references). Actually, in these proposals, links between objects are often assumed to be found automatically by the system. Since in this paper 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/dataspace setting. For example, SPARQL [2], the standard language to query RDF graphs [1], is closely related to database queries on ternary relations [16, 22]; RDF graphs can be viewed as a dataspace model.

   Under the natural assumption that linking between two objects is done in terms of abstract 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 [20, 21]. 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 characterises our associative search queries.

6. Finally, we show that our general theory is workable by applying it to the most common 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 are unnormalized. 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 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.

This paper is organised as follows. Section 2 presents the abstract model of search queries on dataspaces; section 3 presents the application to classical database relations; section 4 extends the abstract model with associative search; section 5 gives the semijoin algebra characterization; and section 6 presents the application to attribute–value dataspaces.

2 Basic model

Basically, we assume given a set $\mathcal{I}$ of abstract data elements called *items*.

Definition 1. An *object* over $\mathcal{I}$ is a finite, nonempty set of items. An *abstract dataspace* over $\mathcal{I}$ is a finite set of objects over $\mathcal{I}$.

Let us see some examples.

- Items could be words over some alphabet, and then objects correspond to documents in the classical boolean model of information retrieval [5, chapter 4.2.3].
- Items could be attribute–value pairs, as in common dataspaces. Dataspace models [11, 12, 17] normally also include links (binary relationships) between objects, but we can represent such links using additional attributes. For example, to make a link *authored_by* between a paper and an author, add an id attribute to the author, and add an attribute *authored_by* to the paper

with the id of the author as value (multiple authors are no problem because objects in dataspaces can contain multiple attribute–value pairs for the same attribute).

- Staying with attribute–value pairs, we could consider dataspaces over objects that contain only attributes from a fixed relation schema, and contain a single value for each such attribute. Then a dataspace is a classical database relation.

We further assume a set $\mathcal{K}$ of predicate names called *abstract keywords*, and a $\mathcal{K}$-structure $(\mathcal{I}, \mathcal{M})$ on $\mathcal{I}$ in which every abstract keyword k is interpreted as a unary predicate $\mathcal{M}(k)$ on $\mathcal{I}$, or, equivalently, a subset $\mathcal{M}(k) \subseteq \mathcal{I}$. The intuition for an item i to be in $\mathcal{M}(k)$ is that i "matches" k.

For example, in the boolean information retrieval model, keywords could be predicates on words that test whether the word contains some fixed string as a substring. On attribute–value pairs, keywords could be predicates on pairs (a, v) that test whether a is some fixed attribute and v is some fixed value.

We identify two special kinds of keywords:

- For each item i we can consider i itself to be a *literal keyword*: the keyword that only matches i itself, i.e., $\mathcal{M}(i) = \{i\}$.

- The *wildcard keyword*, denoted by $\star$, matches all items, i.e., $\mathcal{M}(\star) = \mathcal{I}$.

We now define BSL: a basic boolean search language to query dataspaces. Formally, the syntax of the language depends on $\mathcal{K}$, and the semantics on $\mathcal{M}$, but for simplicity we omit this from our notation.

Definition 2. The expressions of BSL are given by the grammar

$$e \quad ::= \quad k \mid e \; \textsf{and} \; e \mid e \; \textsf{or} \; e \mid e \; \textsf{except} \; e,$$

where k ranges over the keywords in $\mathcal{K}$.

The semantics of these expressions is the following. An expression e can be applied to a dataspace D, resulting in a subset $e(D)$ of D defined as follows:

$$\begin{aligned} k(D) &:= \{o \in D \mid \exists i \in o : i \in \mathcal{M}(k)\} \\ (e_1 \; \textsf{and} \; e_2)(D) &:= e_1(D) \cap e_2(D) \\ (e_1 \; \textsf{or} \; e_2)(D) &:= e_1(D) \cup e_2(D) \\ (e_1 \; \textsf{except} \; e_2)(D) &:= e_1(D) - e_2(D). \end{aligned}$$

Note that the language is a bit redundant as $e_1 \; \textsf{and} \; e_2$ is equivalent to $e_1 \; \textsf{except} \; (e_1 \; \textsf{except} \; e_2)$.

It is important to appreciate the existential nature of the above-defined semantics. Thus, the expression $k_1 \; \textsf{except} \; k_2$ 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. Henceforth, for an object o and a keyword k, we will write $o \models k$ to denote that there exists $i \in o$ such that $i \in \mathcal{M}(k)$. (This notation is a bit sloppy as it does not include $\mathcal{M}$.)

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

Definition 3. A *search query* is a mapping q from dataspaces to dataspaces such that $q(D) \subseteq D$ for each 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 4. Let q be a search query and let $K \subset \mathcal{K}$ be a finite 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 $o_1 \models k$ iff $o_2 \models k$. We denote this by $o_1 \simeq_K o_2$.

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 $o_1 \in q(D_1)$ iff $o_2 \in q(D_2)$.

The above three properties represent three distinctive features of BSL 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 $o \in q(D)$ iff $o \in q(\{o\})$ 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 5. *A search query q is definable in* BSL *if and only if q is K-safe and K-distinguishing, for some finite set $K \subset \mathcal{K}$.*

Proof. The only-if direction is straightforward; for K we take the set of keywords occurring in the BSL expression. The properties of safety and distinguishing are then verified by structural induction (details omitted). For the if-direction, we first observe that each $\simeq_K$ equivalence class C is definable in BSL in the sense that we have an expression e_C such that $o \in C$ iff $o \in e_C(\{o\})$. The only exception is the equivalence class of objects that do not satisfy any keyword in K; but since q is K-safe that equivalence class can be ignored. Since q is K-distinguishing, q can then be defined as the union of all expressions e_C over all C for which $o \in q(\{o\})$ for $o \in C$. We omit the details. □

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

Corollary 6. *If $\mathcal{M}$ includes the wildcard keyword, then a search query is definable in* BSL *iff it is K-distinguishing for some finite K.*

We conclude this section by giving two illustrations of how one can show that certain queries are not definable in BSL.

Example 7. One may wonder if negations of keywords can be expressed: can we retrieve all objects containing an item that does not match some keyword k? The answer in general is no. In proof, suppose $\mathcal{M}$ consists just of a single keyword k, with at least one matching item i and at least one nonmatching item j. In addition, we can also use the wildcard. Then the query "$\neg k$", asking for all objects with an item that does not match k, is not $\{k\}$-distinguishing.[1] Indeed, $\{i\} \simeq_{\{k\}} \{i,j\}$, but $\{i,j\}$ satisfies the query whereas $\{i\}$ does not. Hence, "$\neg k$" is not definable in BSL.

Of course, one can add negated keywords directly to $\mathcal{K}$ and $\mathcal{M}$ to express such queries. One may then wonder whether that is enough to allow already 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, $\mathcal{M}$ contains, besides the wildcard, two keywords k_1 and k_2 and their negations, with items i_1, i_2, j_0, and j, 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. Then the query "$\neg k_1 \wedge \neg k_2$" is not K-distinguishing with $K = \{k_1, k_2, \neg k_1, \neg k_2\}$ and thus not definable in BSL. Indeed, $\{i_1, i_2\} \simeq_K \{j_0, j\}$, but $\{j_0, j\}$ satisfies the query whereas $\{i_1, i_2\}$ does not.

3 Application to classical relations

Let us fix an infinite domain $\mathcal{V}$ of values and a relation schema Σ, i.e., 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. A tuple $t : \Sigma \to \mathcal{V}$ is an an object over $\mathcal{I}$, and a finite relation R over

[1] The attentive reader is assured that $\{k, \star\}$-distinguishing is the same as $\{k\}$-distinguishing.

Σ is a dataspace over $\mathcal{I}$. Note that conversely, not all objects over $\mathcal{I}$ are proper tuples, because in a proper tuple every attribute occurs, with only one value. Hence, not all dataspaces in this setting are proper relations. Let us denote the class of finite relations over Σ by $\mathcal{R}$.

As keywords, we use all literal keywords together with the wildcard. Note that such a literal keyword (a, v), with $a \in \Sigma$ and $v \in \mathcal{V}$, corresponds to the relational selection operator $\sigma_{a=v}$. So, in this setting, the language BSL corresponds to the fragment of the relational algebra consisting of just the three operators of constant selection, union, and difference.

It now turns out that in this setting there is a connection between K-distinguishing search queries and fully C-generic queries. Fully generic queries without constants have been investigated (in a much richer setting than mere selection queries) by Beeri, Milo and Ta-Shma [6, 7]; here we consider the version with constants. Thus, 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 can apply mappings $f : \mathcal{V} \to \mathcal{V}$ to items, objects and dataspaces in the canonical way. 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 $q(f(R)) = f(q(R))$. We establish:

Proposition 8. *Let C be a finite set of constants, and let $K = \{(a, v) \mid a \in \Sigma,\ v \in C\}$. Then a search query q is K-distinguishing on the class $\mathcal{R}$ if and only if q is additive and fully C-generic on $\mathcal{R}$.*

Proof. The only-if direction is based on the fact that for any tuple t and any C-epimorphism f, we have $t \simeq_K f(t)$. For the if-direction, assume $t \simeq_K t'$. We can find a tuple u and C-epimorphisms f and f' such that $f(u) = t$ and $f'(u) = t'$. We then have

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

$\square$

From the above proposition and Corollary 6 we then obtain:

Corollary 9. *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.

4 Associative search

Since BSL queries are additive, BSL cannot define queries that relate objects to other objects; BSL cannot do joins. In dataspace systems, however, we want to be able to retrieve not just all objects that satisfy some query, but also all objects that are related to those objects [11, 12].

To this end, we extend our theory by further assuming a set $\mathcal{L}$ of abstract *link conditions*; we also extend our structure $\mathcal{M}$ to interpret these conditions as binary relationships between objects. So, for each $\lambda \in \mathcal{L}$, we have $\mathcal{M}(\lambda) \subseteq \mathcal{O} \times \mathcal{O}$, where $\mathcal{O}$ is the set of all objects.

We now define our associative search language ASL as an extension of the basic search language BSL with an operator for retrieving related objects:

Definition 10. 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

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

For example, in the boolean information retrieval setting, we might have a link condition $\mathsf{relevant}$, interpreted as the relationship "document o_1 is relevant to document o_2" (computed according to some IR algorithm). Then the expression

$$\mathsf{link}\langle\mathsf{relevant}\rangle(\text{'ICDT' or 'St.Petersburg'})$$

retrieves all documents that are relevant to documents containing the keywords 'ICDT' or 'St.Petersburg'.

There is an obvious connection between ASL and modal logic; indeed, ASL *is* a modal logic. We can make this connection precise by defining a bisimulation notion appropriate for ASL.

Definition 11. 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 the following conditions hold for $n > 0$:

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

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

We then obtain the following lemma known from the model theory of modal logic [15]:

Lemma 12. *Let K be a finite nonempty set of keywords and let L be a finite nonempty set of link conditions. Let $ASL[K, L]$ denote the set of ASL expressions using keywords in K and link conditions in L. 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 $ASL[K, L]$ by an expression with nesting depth of* link *operators at most n.*

2. *q is $\rightleftarrows_n^{K,L}$-invariant on $\mathcal{D}$, i.e., if $(D, o) \rightleftarrows_n^{K,L} (D', o')$ with D and D' in $\mathcal{D}$, then $o \in q(D)$ iff $o' \in q(D')$.*

We will put this lemma to good use later; now, we discuss the more pressing issue of how link conditions should actually be defined. Up to now, link conditions have remained abstract in our formalism, but a flexible search query language should allow them to be expressed in the language itself. In section 6 we will look at the concrete case of attribute–value dataspaces, but here we will look at an intermediate level where link conditions are expressed on the level of, still abstract, relationships between items rather than objects.

So, assume now a set $\mathcal{S}$ of *similarity relation names*, or *simrels* for short. Each simrel $\sim$ is interpreted as a binary relation $\mathcal{M}(\sim) \subseteq \mathcal{I} \times \mathcal{I}$ on items. We now define the following basic class of link conditions, based on simrels and keyword search:

Definition 13. 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:

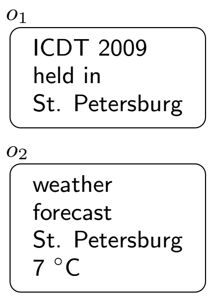
$$\mathcal{M}(k \sim l) = \{(o, o') \in \mathcal{O} \times \mathcal{O} \mid \exists i \in o : \exists j \in o' : \\ i \in \mathcal{M}(k) \text{ and } j \in \mathcal{M}(l) \text{ and } i \sim j\}$$

The intuition behind simlinks is very natural. We search object o on keyword k; we search object o' on keyword l; we compare the two search results and require that they contain a pair of similar items. For a simple example, in the boolean information retrieval setting, we might have a simrel soviet between words such that w_1 soviet w_2 if w_1 is a location name (city name, street name) from the Soviet era, and w_2 is the corresponding post-Soviet name. For example, Leningrad soviet St.Petersburg. Then the expression

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

retrieves all documents containing Soviet versions of locations names mentioned in documents about ICDT. In Section 6 we will see more examples of simlinks.

For the remainder of this paper, we will always use the language ASL with simlinks as link conditions.



(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 D

Figure 1: A dataspace and its relational representation.

5 Semijoin algebra

Acknowledging that search queries are a special kind of database queries, it is 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. The semijoin algebra is the version of the relation algebra where the join operator is replaced by the semijoin operator [20, 21].

Since the relational algebra works on relations, we need a relational representation of a dataspace. So we will view a dataspace D as the binary relation $\{(o, i) \mid o \in D \text{ and } i \in o\}$ over the relation schema $\{\mathsf{id}, \mathsf{item}\}$. The objects in the id-column of this relation are regarded as object identifiers; the relational algebra is not able to peek inside the objects in any other way than working with the item column. An example is shown in Figure 1.

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 id's) are used as semijoin conditions. The relation symbol T (for 'Table') stands for the input dataspace, viewed as a relation over the schema $\{\mathsf{id}, \mathsf{item}\}$. Every expression has an output schema that is either empty, the schema $\{\mathsf{id}, \mathsf{item}\}$ itself, or one of the unary schemas $\{\mathsf{id}\}$ or $\{\mathsf{item}\}$ (we do not have renaming). Expressions must be well-typed, e.g., we only allow the union of two relations over the

$$\frac{}{T : \{\mathsf{id}, \mathsf{item}\}} \qquad \frac{E : \Sigma \qquad \mathsf{item} \in \Sigma \qquad k \in \mathcal{K}}{\sigma_k(E) : \Sigma}$$

$$\frac{E : \Sigma \qquad \Delta \subseteq \Sigma}{\pi_\Delta(E) : \Delta} \qquad \frac{E_1 : \Sigma \qquad E_2 : \Sigma}{E_1 \cup E_2 : \Sigma}$$

$$\frac{E_1 : \Sigma \qquad E_2 : \Sigma}{E_1 - E_2 : \Sigma}$$

$$\frac{E_1 : \Sigma_1 \qquad E_2 : \Sigma_2 \qquad \Sigma_1 \cap \Sigma_2 = \{\mathsf{id}\}}{E_1 \ltimes E_2 : \Sigma_1}$$

$$\frac{\begin{matrix} E_1 : \Sigma_1 \\ E_2 : \Sigma_2 \qquad \Sigma_1 \cap \Sigma_2 = \{\mathsf{item}\} \qquad \sim\, \in \mathcal{S}\end{matrix}}{E_1 \underset{\sim}{\ltimes} E_2 : \Sigma_1}$$

$$\frac{E_1 : \{\mathsf{id}, \mathsf{item}\} \qquad E_2 : \{\mathsf{id}, \mathsf{item}\} \qquad \sim\, \in \mathcal{S}}{E_1 \underset{=,\sim}{\ltimes} E_2 : \{\mathsf{id}, \mathsf{item}\}}$$

Figure 2: Syntax of SA.

same schema. The full syntax of semijoin algebra (abbreviated SA) expressions, together with the derivation of their output schemas, is now given in Figure 2. The notation $E : \Sigma$ 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:

$$\sigma_k(R) := \{t \in R \mid t(\mathsf{item}) \in \mathcal{M}(k)\}$$

$$R_1 \ltimes R_2 := \{t_1 \in R_1 \mid \exists t_2 \in R_2 : t_2(\mathsf{id}) = t_1(\mathsf{id})\}$$

$$R_1 \underset{\sim}{\ltimes} R_2 := \{t_1 \in R_1 \mid \exists t_2 \in R_2 : (t_1(\mathsf{item}), t_2(\mathsf{item})) \in \mathcal{M}(\sim)\}$$

$$R_1 \underset{=,\sim}{\ltimes} R_2 := \{(o, i) \in R_1 \mid \exists j : (o, j) \in R_2 \text{ and } (i, j) \in \mathcal{M}(\sim)\}$$

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.

The following is now expected:

Proposition 14. *Each ASL query (with simlinks) can be expressed in SA.*

Proof. Here is a straight syntactic translation:

$$\begin{aligned}\mathsf{SA}[k] &:= \sigma_k(T)\\ \mathsf{SA}[e_1 \text{ or } e_2] &:= \pi_{\mathsf{id}}\mathsf{SA}[e_1] \cup \pi_{\mathsf{id}}\mathsf{SA}[e_2]\\ \mathsf{SA}[e_1 \text{ except } e_2] &:= \pi_{\mathsf{id}}\mathsf{SA}[e_1] - \pi_{\mathsf{id}}\mathsf{SA}[e_2]\end{aligned}$$

$$\mathsf{SA}[\mathsf{link}\langle k \sim l\rangle\, e] := \\ \pi_{\mathsf{id}}(\mathsf{SA}[k] \ltimes_{\sim} \pi_{\mathsf{item}}(\mathsf{SA}[l] \ltimes \pi_{\mathsf{id}}\mathsf{SA}[e]))$$

□

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. For example, the SA expression $\pi_{\mathsf{id}}(T - \sigma_k(T))$ defines the negated keyword $\neg k$, i.e., the query $q(D) = \{o \in D \mid \exists i \in o : i \notin \mathcal{M}(k)\}$. We have already seen in Example 7 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. Syntactically, we move from $\mathcal{K}$ to $\mathcal{K}^*$ which is the smallest set of keywords containing $\mathcal{K}$ and closed under the syntactic operators $\neg$ and $\vee$; semantically, we extend the interpretation $\mathcal{M}$ of $\mathcal{K}$ to an interpretation $\mathcal{M}^*$ of $\mathcal{K}^*$ in the natural way. We can then revise Proposition 14 as follows:

Proposition 14 revised. *Every ASL query over $\mathcal{M}^*$ can be defined in SA over $\mathcal{M}$.*

Proof. The syntactic translation from the proof of Proposition 14 can be extended as follows: (φ and ψ stand for boolean combinations of keywords)

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

□

Note that a boolean closed set of keywords automatically includes a wildcard keyword, in the form of $k \vee \neg k$.

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

Proposition 15. *Each of the following SA queries is not definable in ASL, even over the boolean closure of keywords:*

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

More precisely, for each of these queries there exists a simple interpretation $\mathcal{M}$ of the keywords k, l and m and the simrel $\sim$ such that the query is not definable in ASL over $\mathcal{M}^*$.

Note that the above four expressions 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 ASL by showing that they are not bisimulation invariant, then invoking Lemma 12.

Proof of Proposition 15. For the query q_1 expressed by E_1, we put $\mathcal{M}(k) = \{a_1, a_2\}$ and $\mathcal{M}(l) = \{b_1, b_2\}$, with $a_1 \sim b_1$ and $a_2 \sim b_2$. Now consider $D = \{o\}$ and $D' = \{o_1, o_2\}$ with $o = \{a_1, b_1\}$; $o_1 = \{a_1, b_2\}$; and $o_2 = \{a_2, b_1\}$. Then (D, o) and (D', o_1) are bisimilar (n-bisimilar for any n). Yet, $o \in q_1(D)$ whereas $o_1 \notin q_1(D')$.

For the query q_2 expressed by E_2, we use items a, a', and b with $\mathcal{M}(k) = \{b\}$ 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 $o'_1 = \{a'\}$. Then (D, o_1) is bisimilar to (D', o'_1), but $o_1 \in q_2(D)$ whereas $o'_1 \notin q_2(D')$.

For the query q_3 expressed by E_3, we put $\mathcal{M}(k) = \{a\}$; $\mathcal{M}(l) = \{b, b_1, b_2\}$; and $\mathcal{M}(m) = \{c\}$, 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 $o'_2 = \{b_1, b_2, c\}$. Then (D, o_1) is bisimilar to (D', o_1), 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 $\mathcal{M}(k) = \{a\}$, and $\sim$ is 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 $o'_1 = \{b\}$. Then (D, o_1) is bisimilar to (D', o'_1), but $o_1 \in q_4(D)$ whereas $o'_1 \notin q_4(D')$. □

The above proposition 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 $\{\mathsf{item}\}$, so we conclude we must always project the result of $\ltimes_\sim$ on $\{\mathsf{id}\}$. Moreover, E_4 suggests that we should not allow projection on $\{\mathsf{item}\}$ altogether, except of course to allow for a semijoin $\ltimes_\sim$ to be applied. We thus arrive at the following fragment of SA, which we denote by $\mathrm{SA}^{\mathrm{search}}$:

Definition 16. The fragment $\mathrm{SA}^{\mathrm{search}}$ 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:

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

We now establish the connection between associative search and the semijoin algebra:

Theorem 17. *A search query is definable in* $\mathrm{SA}^{\mathrm{search}}$ *if and only if it is definable in ASL (with simlinks) over the boolean closure of keywords.*

Proof. The if-direction follows from the observation that the translation from ASL to SA given in the proof of Proposition 14 stays within the fragment $\mathrm{SA}^{\mathrm{search}}$.

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

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

- if the output schema of E is $\{\mathsf{id}, \mathsf{item}\}$, then $E(D)$ equals
$$\bigcup_{(e,k)\in\chi_E} \{(o, i) \mid o \in e(D),\ i \in o,\ i \in \mathcal{M}(k)\};$$
- if the output schema of E is $\{\mathsf{id}\}$, then $E(D)$ 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 $\{\mathsf{id}\}$, and thus we can define q in ASL as $(e_1 \,\mathsf{and}\, k_1)\,\mathsf{or}\ldots\mathsf{or}\,(e_n \,\mathsf{and}\, k_n)$ where $\chi_E = \{(e_1, k_1), \ldots, (e_n, k_n)\}$.

We construct χ_E by induction as follows:

$$\begin{aligned}
\chi_T &:= \{(\star, \star)\} \\
\chi_{\sigma_l(E)} &:= \{(e, k \wedge l) \mid (e, k) \in \chi_E\} \\
\chi_{\pi_{\mathsf{id}}(E)} &:= \{(e \,\mathsf{and}\, k, \star) \mid (e, k) \in \chi_E\} \\
\chi_{E_1 \cup E_2} &:= \chi_{E_1} \cup \chi_{E_2}
\end{aligned}$$

$$\begin{aligned}
\chi_{E_1 - E_2} := \{&(e_1 \,\mathsf{except}\, e_2, k_1), (e_1, k_1 \wedge \neg k_2) \mid \\
&(e_1, k_1) \in \chi_{E_1},\ (e_2, k_2) \in \chi_{E_2}\}
\end{aligned}$$

$$\begin{aligned}
\chi_{\pi_{\mathsf{id}}(E_1 \ltimes_\sim \pi_{\mathsf{item}} E_2)} := \{&(e_1 \,\mathsf{and}\, \mathsf{link}\langle k_1 \sim k_2\rangle\, e_2, \star) \mid \\
&(e_1, k_1) \in \chi_{E_1},\ (e_2, k_2) \in \chi_{E_2}\}
\end{aligned}$$

$$\begin{aligned}
\chi_{E_1 \ltimes E_2} := \{&(e_1 \,\mathsf{and}\, e_2 \,\mathsf{and}\, k_2, k_1) \mid \\
&(e_1, k_1) \in \chi_{E_1},\ (e_2, k_2) \in \chi_{E_2}\}
\end{aligned}$$

□

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 $i \sim_2 j$. We can show (proof omitted) that such conjunctive join conditions bring us outside the semijoin algebra. Nevertheless, it is still possible to define a fragment of the relational algebra and prove a characterization along the lines of Theorem 17. We omit the details.

6 Attribute–value dataspaces

Let us now apply our abstract theory to the concrete setting of dataspaces as they have been investigated in the literature [11, 12, 17]. 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.

Dataspace models in the literature give objects 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 id's 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.

The next question is what would be an interesting universe $\mathcal{K}$ of keywords. Surely we already want all literal keywords (a, v) to be present (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':[2]

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

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 $\mathsf{Heineken}$, and $\neg\mathsf{likes}\colon \mathsf{Heineken}$ retrieves objects containing $\mathsf{Heineken}$ 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 $\mathsf{Belgium}$ 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 (AV) pairs.

Definition 18. An *AV keyword* is a pair of one of the following forms:

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

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

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

are interpreted as follows:

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

Note that $\neg\emptyset$ plays the role of a wildcard $\star$ on attributes or values, and then $(\star\colon \star)$ plays the role of the wildcard on pairs.

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

Proposition 19. *BSL over AV keywords is equivalent to BSL over the boolean closure of AV keywords.*

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. □

Since we have the wildcard, Corollary 6 applies, so we know that in the AV setting, BSL defines exactly all search queries that are K-distinguishing for some finite set K of AV keywords. Recall that a search query q is K-distinguishing if it is invariant under the equivalence $\simeq_K$ on objects (Definition 4). In the AV 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 20. Let W be a finite set of attributes and values. Let $\diamond$ be a "blank value", that is an arbitrary element not in W. For an attribute or value x, define

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

We can extend blank_W to AV pairs, objects, and dataspaces in the canonical, pointwise manner. Two objects o_1 and o_2 are called *W-equivalent* if $\mathrm{blank}_W(o_1) = \mathrm{blank}_W(o_2)$. 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 $o_2 \in q(D_2)$.

Proposition 21. *A search query is K-distinguishing for a finite set of AV keywords K, 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 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. □

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

Definition 22. An *eqrel* (short for equality relation) is one of the three following simrels on AV pairs:

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

For example, the following is an expression in the associative search language ASL over AV keywords and eqrel simlinks:

link⟨(name: $\star$) eq-val (author: $\star$)⟩
(published in: 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).)

In other dataspace models based on attribute–value pairs [11, 12], 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 graph of objects. Using eqrel simlinks as we do, an explicit graph model is redundant. Indeed, as just illustrated in the above example, named edges can easily be represented using id attributes for objects and "pointer" attributes having these ids as values.

We next show that the abstract bisimulation Lemma 12 can well be applied in the present AV setting as an aid to understand the limits of expressive power of ASL. 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 in the AV setting; we prove it is not:

Proposition 23. *$q_{a=b}$ is not definable in ASL over eqrel simlinks and AV keywords.*

The proof shows that $q_{a=b}$ is not bisimulation invariant relative to any finite set of AV keywords and eqrel simlinks, then invokes Lemma 12. The proof is rather intricate and has been relegated to the Appendix.

We conclude this section by returning to the equivalence between ASL and the semijoin algebra. Working with abstract dataspaces in Section 5, we have defined the semijoin algebra working on dataspaces represented as binary relations over the schema $\{\mathsf{id}, \mathsf{item}\}$. While the equivalence of ASL and $\mathrm{SA}^{\mathrm{search}}$ (Theorem 17) can be directly applied to the AV setting, it is not so natural to store AV pairs in a single item column. It is more natural to represent AV dataspaces as sets of triples, i.e., as *ternary* relations over the schema $\{\mathsf{id}, \mathsf{attr}, \mathsf{val}\}$. Also the RDF query language SPARQL works over such ternary relations [2, 16, 22]; RDF graphs can also be viewed as a dataspace model.

So, it is worthwhile to define an alternative to $\mathrm{SA}^{\mathrm{search}}$ 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 24. The fragment $\mathrm{SA}^{\mathrm{AV}}$ of the semijoin algebra, defined on relations $T : \{\mathsf{id}, \mathsf{attr}, \mathsf{val}\}$, is defined by the following grammar:

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

where c ranges over attribute and value constants, and α is either $\{\mathsf{attr}\}$, $\{\mathsf{val}\}$, or $\{\mathsf{attr}, \mathsf{val}\}$.

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

We have the following analog of Theorem 17:

Theorem 25. *A search query is definable in ASL over AV keywords and eqrel simlinks, if and only if it is definable in $\mathrm{SA}^{\mathrm{AV}}$, where we regard a dataspace D as a ternary relation $\{(o, a, v) \mid o \in D$ and $(a, v) \in o\}$.*

Proof. The only-if direction is similar to the proof of Theorem 17. AV keywords are expressed using combinations of constant selections using union and difference. Simlinks based on $\mathsf{eq\text{-}attr}$, $\mathsf{eq\text{-}val}$, or eq are expressed using semijoin with projection on the right (the set α in the syntax definition) equal to $\{\mathsf{attr}\}$, $\{\mathsf{val}\}$, or $\{\mathsf{attr}, \mathsf{val}\}$, 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 19, we can simulate them in the language. Proposition 19 is only formulated for BSL, but the same argument holds for ASL, because link distributes over a disjunction of keywords used in a simlink: $\mathsf{link}\langle(\varphi_1 \vee \varphi_2) \sim \psi\rangle(e) = \mathsf{link}\langle\varphi_1 \sim \psi\rangle(e)$ or $\mathsf{link}\langle\varphi_2 \sim \psi\rangle(e)$. Semijoins are translated to simlinks using an eqrel depending on α: if $\alpha = \{\mathsf{attr}\}$ we use $\mathsf{eq\text{-}attr}$, if $\alpha = \{\mathsf{val}\}$ we use $\mathsf{eq\text{-}val}$, and if $\alpha = \{\mathsf{attr}, \mathsf{val}\}$ we use eq. $\square$

7 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. (One of us has studied unrestricted, i.e., relationally complete querying of dataspace-like databases earlier [19].)

We have 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 [12], we have focused on selection queries, i.e., queries that always return a subset of the original objects. Current approaches to keyword search on structured databases [14, we give just one recent reference] 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, given their close similarity to AV dataspaces. It might be possible to specialize our semijoin algebra fragment $\mathrm{SA}^{\mathrm{AV}}$ into a fragment of SPARQL, which is some kind of specialized relational algebra for ternary relations (RDF triples) [13, 22].

References

[1] RDF primer. W3C Recommendation, February 2004.

[2] SPARQL query language for RDF. W3C Recommendation, January 2008.

[3] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[4] A.V. Aho and J.D. Ullman. Universality of data retrieval languages. In *Conference Record, 6th ACM Symposium on Principles of Programming Languages*, pages 110–120, 1979.

[5] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley, 1999.

[6] C. Beeri, T. Milo, and P. Ta-Shma. On genericity and parametricity. In *Proceedings 15th ACM Symposium on Principles of Database Systems*, pages 104–116, 1996.

[7] C. Beeri, T. Milo, and P. Ta-Shma. Towards a languages for the fully generic queries. In S. Cluet and R. Hull, editors, *Database Programming Languages*, Lecture Notes in Computer Science, pages 239–259. Springer, 1997.

[8] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic.* Cambridge University Press, 2001.

[9] P. Blackburn, J. van Benthem, and F. Wolter, editors. *Handbook of Modal Logic.* Elsevier, 2007.

[10] A. Chandra and D. Harel. Computable queries for relational data bases. *Journal of Computer and System Sciences*, 21(2):156–178, 1980.

[11] J.-P. Dittrich and M.A. Vaz Salles. iDM: A unified and versatile data model for personal dataspace management. In *Proceedings 32nd International Conference on Very Large Data Bases*, pages 367–378, 2006.

[12] X. Dong and A. Halevy. Indexing dataspaces. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 43–54, 2007.

[13] George H.L. Fletcher. An algebra for basic graph patterns. Presented at the workshop

on Logic in Databases, Rome, Italy, May 2008.

[14] K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword proximity search in complex data graphs. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 927–940, 2008.

[15] V. Goranko and M. Otto. Model theory of modal logic. In Blackburn et al. [9], chapter 5.

[16] C. Gutierrez, C. Hurtado, and A. Mendelzon. Foundations of semantic web databases. In *Proceedings 23rd ACM Symposium on Principles of Database Systems*, pages 95–106, 2004.

[17] A. Halevy, M. Franklin, and D. Maier. Principles of dataspace systems. In *Proceedings 25th ACM Symposium on Principles of Database Systems*, pages 1–9, 2006.

[18] H.V. Jagadish et al. Making database systems usable. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 13–24, 2007.

[19] M. Jain, A. Mendhekar, and D. Van Gucht. A uniform data model for relational data and meta-data query processing. In *Advances in Data Management '95*, pages 146–165. Tata McGraw-Hill, 1995.

[20] D. Leinders, M. Marx, J. Tyszkiewicz, and J. Van den Bussche. The semijoin algebra and the guarded fragment. *Journal of Logic, Language and Information*, 14:331–343, 2005.

[21] D. Leinders and J. Van den Bussche. On the complexity of division and set joins in the relational algebra. *Journal of Computer and System Sciences*, 73(4):538–549, 2007.

[22] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. In *The Semantic Web: Proceedings ISWC*, volume 4273 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 2006.

[23] J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume II. Computer Science Press, 1989.

Appendix

Proof of Proposition 23. Consider an arbitrary expression e in ASL. Let K be the finite set of keywords mentioned in e (be it as an expression or in a simlink), let

$$L = \{k \sim l \mid k, l \in K,\ \sim \text{ an eqrel}\},$$

and let n be the nesting depth of link expressions in e. In this proof we will use $\mathcal{W}$ to denote the set $\Sigma \cup \mathcal{V}$ of all attributes and values.

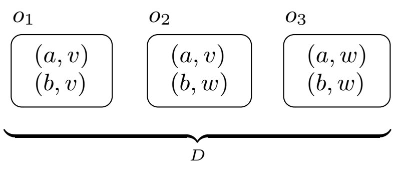
To see that e cannot define $q_{a=b}$, first define, for every $M \subseteq K$, the set $\llbracket M \rrbracket$ of pairs matched by all keywords in M and none of the keywords in $K - M$:

$$\llbracket M \rrbracket := (\mathcal{W} \times \mathcal{W}) \cap \bigcap_{k \in M} \mathcal{M}(k) - \bigcup_{k' \in K - M} \mathcal{M}(k').$$

Clearly, $\{\llbracket M \rrbracket \mid M \subseteq K\}$ forms a partition of $\mathcal{W} \times \mathcal{W}$. In particular, $\mathcal{W} \times \mathcal{W} = \bigcup_{M \subseteq K} \llbracket M \rrbracket$. Since the subset $\{(a,u) \mid u \in \mathcal{W}\}$ of $\mathcal{W} \times \mathcal{W}$ 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 $\llbracket M^a \rrbracket \cap \{(a,u) \mid u \in \mathcal{W}\}$ is infinite. Then let $\{M_1^b, \ldots, M_m^b\} = \{M \subseteq K \mid \exists u \in \mathcal{W}(b,u) \in \llbracket M \rrbracket\}$. Since $\{u \mid (a,u) \in \llbracket M^a \rrbracket\}$ is infinite, since $\mathcal{W} = \bigcup_{m=1}^{n} \{u \mid (b,u) \in \llbracket M_i^b \rrbracket\}$, and since m is finite, there must exist some $1 \leq i \leq m$ such that

$$\{u \mid (a,u) \in \llbracket M^a \rrbracket\} \cap \{u \mid (b,u) \in \llbracket M_i^b \rrbracket\}$$

is infinite. Then fix v, w in this intersection. By construction, $(a,v), (a,w) \in \llbracket M^a \rrbracket$ and $(b,v), (b,w) \in \llbracket M_i^b \rrbracket$, i.e., (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 12, 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 $(o_1, p_1) \in \mathcal{M}(\lambda)$. We show that there exists $p_2, p_3 \in D$ such that (1) $(o_2, p_2) \in \mathcal{M}(\lambda)$ and $(D, p_1) \rightleftarrows_{n-1}^{K,L} (D, p_2)$ and (2) $(o_3, p_3) \in \mathcal{M}(\lambda)$ and $(D, p_1) \rightleftarrows_{n-1}^{K,L} (D, p_3)$.

- Case $\lambda = k$ eq l. Then $o_1 \cap \mathcal{M}(k) \cap p_1 \cap \mathcal{M}(l)$ is non-empty. 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 $o_2 \cap \mathcal{M}(k) \cap p_2 \cap \mathcal{M}(l)$ (which is hence non-empty). Moreover, since $\rightleftarrows$ is an equivalence relation and $p_1 = p_2$, also $(D, p_1) \rightleftarrows_{n-1}^{K,L} (D, p_2)$. To see that it suffices to take $p_3 := o_3$, observe that by construction also $(a, w) \in o_3 \cap \mathcal{M}(k) \cap p_3 \cap \mathcal{M}(l)$ 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 $(D, o_1) \rightleftarrows_{n-1}^{K,L} (D, o_2)$ and $(D, o_1) \rightleftarrows_{n-1}^{K,L} (D, o_3)$ (by induction hypothesis); since hence also $(D, o_2) \rightleftarrows_{n-1}^{K,L} (D, o_3)$ (by transitivity of $\rightleftarrows$); and since $(D, o_3) \rightleftarrows_{n-1}^{K,L} (D, o_3)$ (by reflexivity of $\rightleftarrows$), certainly $(D, p_1) \rightleftarrows_{n-1}^{K,L} (D, p_3)$. (2) When (b, v) is in the intersection, as symmetric argument shows that it suffices to take $p_2 := o_2$ and $p_3 := p_1$.

- Case $\lambda = k$ eq-attr l. Then $\pi_1(o_1 \cap \mathcal{M}(k)) \cap \pi_1(p_1 \cap \mathcal{M}(l))$ is non-empty. As such, either a is in the intersection, or b is. In both event, 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 $\pi_2(o_1 \cap \mathcal{M}(k)) \cap \pi_2(p_1 \cap \mathcal{M}(l))$ is non-empty. Since v is the only word in $\pi_2(o_1)$, $\pi_2(o_1 \cap \mathcal{M}(k)) \cap \pi_2(p_1 \cap \mathcal{M}(l)) = \{v\}$. There are four possibilities why this is so:

  1. $(a, v) \in o_1 \cap \mathcal{M}(k)$ and $(a, v) \in p_1 \cap \mathcal{M}(l)$;
  2. $(a, v) \in o_1 \cap \mathcal{M}(k)$ and $(b, v) \in p_1 \cap \mathcal{M}(l)$;
  3. $(b, v) \in o_1 \cap \mathcal{M}(k)$ and $(a, v) \in p_1 \cap \mathcal{M}(l)$;
  4. $(b, v) \in o_1 \cap \mathcal{M}(k)$ and $(b, v) \in p_1 \cap \mathcal{M}(l)$.

  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. □