UNIVERSITY OF OSLO
Department of Informatics

A Formal Comparison of ADT-based Dimensional Query Languages

Bjørn Skjellaug

Research Report No 276

ISBN 82-7368-222-6
ISSN 0806-3036

September 1999

A Formal Comparison of ADT-based Dimensional Query Languages

Bjørn Skjellaug
SINTEF Telecom and Informatics *and* Dep. of Informatics, University of Oslo,
P.O.Box 124, Blindern, Forskningsveien 1, N–0314 Oslo, Norway,
Phone: +47 22067300, Fax: +47 22067350, bjornsk@ifi.uio.no

Abstract

This paper investigates and formally compares the expressive power of dimensional (i.e., spatial, temporal, and spatio-temporal) query languages, where the dimensional extensions are supported in terms of ADTs (abstract data types). There are basically two approaches to the design of dimensional ADT extended query languages. One approach, by definition, adds semantics by interpreting an ADT attribute value associated with a database fact as an *intrinsic* (i.e., built-in) relationship with an underlying space. The other approach treats ADT attribute values as conventional attributes, where the dimension semantics (and space) associated with a database fact is an *extrinsic* property and controlled fully by the user.

The comparison framework is based on the relational algebra (RA) and a single ADT extension to RA. Two comparison criteria of semantic equivalence also are defined. The one criterion of *strict equivalent* expressions imposes equal results, whereas the other (relaxed) criterion of *snapshot equivalent* expressions imposes equal snapshot results. For the strict criterion a certain class of intrinsic ADT extended languages is semantically richer than the set of corresponding expressions of a pure (i.e., extrinsic) ADT extended language. This is due to the properties of the built-in dimension support. For the relaxed criterion the same intrinsic language class is shown snapshot equivalent with corresponding expressions of the pure ADT extended language class. However, there is a class of expressions which relates database facts of non-intersecting subspaces, that is expressible only by the pure ADT language. In general, and despite differences, one language approach is not found strictly superior to the other. Rather, practically, the findings indicate multi-approach designs for user-level oriented query languages. Moreover, the findings also informally indicate that by extending the framework, e.g., allowing multiple orthogonal dimension ADTs, more involved problems arise, such as a kind of indeterminism of pure ADT extended languages, i.e., user-choices influence results of otherwise orthogonal dimensions.

1 Introduction

This paper is concerned with extensions of query languages which address data referenced by an underlying dimensional space, such as supported by spatial, temporal, and spatio-temporal query languages, and where these references are defined in terms of abstract data types (ADTs). General purpose commercial database systems, such as Informix, Oracle, and DB2, have to some degree support for spatial and temporal data management based on ADT extensions. Thus, experiences and approaches achieved by the database research community should be of both practical importance and interest in developing such systems.

Especially, spatial and temporal database research have adopted different principles in query language design, e.g., see [9, 17] and [4, 14, 19], respectively. The principle differences are shown by the fact that a temporal query language typically redefines their underlying algebra to become temporal, and, thereby, make dimension semantics an intrinsic property of the algebra, and the fact that a spatial query language only adopts the ADT extension, but, leaves the dimension semantics as an extrinsic property. Er-

wig et al. explore in their paper [7] the expressive power of a selected set of spatio-temporal data models, but, there exists, to our knowledge, no formal study of the relative expressiveness of query languages based on extrinsic versus intrinsic ADT dimension semantics.

Spatial database research has focused mostly on spatial datatypes, i.e., their structures, operations and semantics, including system internal indexing structures, etc. (e.g., [18]). Integration of spatial dimension semantics with the logical data model and query language has not yet been fully addressed. Thus, a spatial attribute is treated analogously to other property data comprising a database fact. Let a sample spatial database which captures information about buildings and estates illustrate this point:

Determine each building spatially associated with an estate.

```
SELECT e.number, b.number
FROM   Estates e, Buildings b
WHERE  CONTAINS(e.region, b.location);
```

The spatial semantics implied by the above query is specified by the user. For example, the "spatial join" is formulated as an old-style join (e.g., cf. [15]) followed by a selection criteria based on a user-specified spatial predicate. According to Güting [9]: *"Strictly speaking, there is no such thing as a spatial selection..."*, and, furthermore, *"...Similar to a spatial selection, a spatial join is a join that compares any two objects with a predicate according to their spatial attributes"*—meaning that the spatial predicate is user-specified.

Temporal database research, on the other hand, has focused on making existing query languages temporal by redefining the algebra with built-in dimension semantics (e.g., [8, 22]). Hence, a conventional (i.e., a snapshot) query is only a special case of a temporal query restricted to only consider the current database state. The temporal semantics is an intrinsic property and given by valid time and/or transaction time dimensions, i.e., managing when a fact is true in the modeled reality, and/or when it is current in the database, respectively [11]. A join of an intrinsic ADT extended language is a built-in natural dimensional join, which, by definition, combines only those operands tuples which have intersecting references to the underlying dimension space. In that sense, and contradictory to the above claim of Güting, there are dimensional selections and dimensional joins, also for spatial ADT extensions, e.g., cf. STSQL ([2]).

A comparison framework is given by extending Codd's relational model [5] with a single 1-dimensional ADT (i.e., an interval or a line segment ADT), over which four algebras are defined, and where each algebra extends the relational algebra (RA) in a particular way. There are two pure (i.e., extrinsic) ADT extended algebras, where one only incorporates new data types, and the other extends with unfolding and folding operators to obtain a point-based, but, still user-controlled interpretation of database facts. They are termed ADT^{P} and $\mathrm{ADT}^{U/F}$, respectively; The two intrinsic ADT extended algebras, where one is based on the property of *snapshot reducibility* ([20]), and where the other goes beyond this notion and combines it with the ability to add user-specified dimension semantics. They are termed the SR and SEQ algebras, respectively.

The comparison defines two criteria of expressive power in terms of semantic equivalent expressions. Expressions are said to be *strict equivalent* if they yield equal results, and expressions are said to be *snapshot equivalent* (SE) [11] if they yield equal snapshot results, i.e.,

when sliced at an arbitrary snapshot. Different properties of the algebras are defined to show how the algebras satisfy the above comparison criteria. First, an ADT extended model relates a database fact with either a point-based dimension semantics, or a region-based dimension semantics, e.g., a interval-based model [3]. An algebra must reflect this distinction, even though that the representation of the dimension value is the same for both algebras. For example, an interval is only a syntactic shorthand for individual reference points of a *point-based* (PB) algebra, whereas it is a reference value in its own right of a *dimension value preserving* (DP) algebra. Moreover, we also define the properties of *dimension parameter expressions* (DPE) as user-specified expressions, and, finally, *inter-* and/or *intra-subspace relationships* of database facts, i.e., dimension references of expressions which address the ability of an algebra in combining (through the Cartesian Product) database facts according to their dimension associations.

Thus, based on the above properties the rational behind the strict criterion is to expose semantic differences and similarities of "corresponding" algebraic expressions. The other criterion is defined to show whether the differences in strict equivalences are eliminated by comparing on snapshot equivalence, i.e., to expose correspondences of some other well-defined semantic notion. For both criteria we also investigate whether one algebra subsume another algebra, i.e., whether the former algebra semantically support all the expressions of the latter algebra, but not necessarily vice versa. Finally, we discuss informally and briefly issues concerning extending the comparison framework.

The paper is structured as follows: First, the algebraic framework and the properties of the comparison is given in Section 2. Section 3 formally defines the algebras and summaries their properties, and Section 4 gives the comparison. Finally, Section 5 concludes the paper.

2 Framework of the Comparison

2.1 Data Structure and Algebra Basics

Codd's relational model [5] is extended in the following way: A relation scheme, $R = (A_1, \ldots, A_n)$, is given by a relation name and a list of attribute names, but where one of the attributes names A_i, $1 \leq i \leq n$, is the dimension attribute, written $A_i = D$, and A is a shorthand for the list $R \setminus D$. An $r(R)$, or simply r, denotes a relation of the scheme R. For a tuple $t \in r(R)$, $t[A]$ and $t[D]$ denotes the the list of A attribute values and the single dimension attribute value of t, respectively. Since D is an interval (line segment) ADT, the $t[D]^s$ and $t[D]^e$ denote the respective begin and end points of $t[D]$. Thus, tuples $t_1 \in r_1(R)$ and $t_2 \in r_2(R)$ are *value equivalent*, if $t_1[A] = t_2[A]$ [11]. The syntax of the corresponding RA language is given by the following coarse set of BNF productions:

$$
\begin{array}{rcl}
RA & ::= & \bot\, EXP\, \top \\
EXP & ::= & r \mid \sigma_P(EXP) \mid \pi_X(EXP) \mid \\
 & & EXP_1 \cup EXP_2 \mid \xi_{\langle X,f\rangle}(EXP) \mid \\
 & & EXP_1 \times EXP_2 \mid EXP_1 \setminus EXP_2 \mid
\end{array}
$$

The non-terminal RA symbolizes a full algebraic expression, and the corresponding right hand side EXP has a start ($\bot$) and an end ($\top$) symbol. The aggregate operator, ξ, equals the definition of *aggregate formation* by Klug [12]. An aggregate is formed based on a list X denoting a (possible empty) list of grouping attributes, and an aggregate function f denoting a particu-

lar aggregation, such as sum, count, min, max, etc., over a specified column associated with an input relation, e.g., $\max_3(r)$ denotes the maximum value of the third column of a relation r. Thus, the function type of f is defined as a mapping from the set of relations to a scalar domain. The rest is standard relational algebraic constructs.

2.2 Dimension Semantics Support

The following introduces the different dimensional properties of an algebra. Initially, let the below example illustrate the semantics of dimension intrinsic expressions. The following two queries are issued over the sample database recording employee and department histories, respectively:

1) *Determine the (periodic) salary pay-outs for each department over all times.*

2) *Determine the employees who have not been a department manager during some period.*

EMP:

name	dept	sal	D
Pete	d_1	10k	[1985-90]
Ann	d_1	15k	[1988-97]

DEPT:

mng	id	D
Ann	d_1	[1991-95]

RESULT of 1)

dept	sum_3	D
d_1	10k	[1985-87]
d_1	25k	[1988-90]
d_1	15k	[1991-97]

RESULT of 2)

name	dept	D
Pete	d_1	[1985-89]
Ann	d_1	[1988-90]
Ann	d_1	[1996-97]

Queries 1) and 2) above are dimensional aggregate formation and dimensional difference, respectively. Notice that both results automatically accounts for periodic changes. Now, let the above queries be expressed in STSQL [2] by 1) and 2) below, respectively:

```
1) REDUCIBLE (D) AS D
     SELECT dept, SUM(sal)
     FROM EMP GROUP BY dept;

2) REDUCIBLE (EMP.D, DEPT.D) AS D
     SELECT name, dept FROM EMP
     EXCEPT SELECT * FROM DEPT;
```

The `REDUCIBLE` flag, which is a STSQL construct, imposes a dimensional query over the referenced ADT dimension `D`. Thus, the flag implies the deployment of an underlying dimensional algebra. Note, however, that by skiping the flag in the above expression STSQL simply would evaluate the query as a pure SQL–92 query, i.e., only involving the current database state. The "bodies" of the above queries are pure SQL–92 queries, which show, when the flag is omitted, the relationship with the RA semantics.

This leads to the aforementioned notion of *snapshot reducibility* [20] of an algebra or a query language. But first we define the notion of *snapshot equivalence*[11] of relations. Concentually, a dimensional database $\mathbf{D}$ may be viewed as a sequence of snapshots, $\mathbf{D} = \langle \ldots, D_0, D_1, D_2 \ldots \rangle$, where each snapshot is related with, or indexed by, a distinct point [4]. This view is utilized by a slice operator, τ_p, which denotes the snapshot of $\mathbf{D}$ at a point p, i.e., $\tau_p(\mathbf{D}) = D_p$. In particular, for a tuple t in r, and p in $t[D]$, $\tau_p(t) = t[A]$, i.e., the A-values at point p. Thus,

Definition 2.1 [11] Two relations r_1 and r_2 are *snapshot equivalent* (SE), $r_1 \overset{se}{\equiv} r_2$, if for all points p, such that

$$\tau_p(r_1) = \tau_p(r_2) \qquad \blacksquare$$

Then, the notion of SE is generalized to account for comparing expressions of a dimensional algebra with its conventional counterpart.

Definition 2.2 ([20]) An algebra (or query language) is *snapshot reducible* (SR) if and only if for all points p, dimensional operators op_X, corresponding to conventional RA operators op^c_X, where X denotes any RA parameter expression, dimensional relations $r_1, \ldots, r_n$, such that

$$\tau_p(op_X(r_1, \ldots, r_n)) = op^c_X(\tau_p(r_1), \ldots, \tau_p(r_n)) \quad \blacksquare$$

The SR-property is based on a point-based comparison of expressions, but there are no requirement what so ever that a SR algebra by definition is a point-based (PB) algebra.

Definition 2.3 Let $\mathcal{A}$ be an algebra, and let $\{r_1, \ldots, r_n\}$ be a set of ADT dimension extended relations. Then, $\mathcal{A}$ is a *point-based* (PB) algebra, iff, for every n-ary operation *op* of $\mathcal{A}$,

$$\begin{aligned}
&\forall\, t, t' \in op(r_1, ..., r_n)\ (\\
&\quad (t \neq t' \wedge t[A] = t'[A]) \Rightarrow \\
&\qquad (disjoint(t[D], t'[D]) \wedge \\
&\qquad \neg meets(t[D], t'[D]) \wedge \\
&\qquad \neg meets(t[D], t'[D]))) \quad \blacksquare
\end{aligned}$$

Definition 2.3 enforces that the input relations are interpreted as populated by tuples each of which has a single point dimension reference (i.e., a D value is a syntactic shorthand), and that the result relations are populated with tuples where no pairs of A-value equivalent tuples intersects or meets on their D values. The two SR query examples below, which illustrates the orthogonality of the SR and PB properties, determine employment histories of departments. The TSQL2 ([21]) query is PB and the STSQL ([2]) query is not PB, respectively. (TSQL2 does not use a flag, and evaluates by default over all states.)

```
TSQL2:  SELECT dept FROM EMP;

STSQL: REDUCIBLE (D) AS D
         SELECT dept FROM EMP;
```

yielding r_1 and r_2, respectively:

$$r_1 = \{\langle d_1, [1985 - 97]\rangle\}$$

$$r_2 = \{\langle d_1, [1985 - 90]\rangle, \langle d_1, [1988 - 97]\rangle\}$$

In general the point-based TSQL2 by definition constructs a single result tuple from each set of qualified (value-equivalent) tuples which forms a maximal chain of contigouos (i.e., a connected set of) points over their D values, cf. [3]. Illustrated by r_1 above. The STSQL query, on the other hand, is semantically richer by being explicit about department d_1 having two employments during two distinct, but overlapping periods, see r_2. (Note, that r_1 and r_2 are SE by Definition 2.1). The STSQL query exhibits the DP property, which associates each resulting database fact with a D value that reflects the semantics of the distinct D values of the input tuples contributing to the construction of the result tuple. Thus, in a DP language the dimensional semantics is given by the D value as a single reference, and not as multiple references by the corresponding set of individual points. On the other hand, a PB language only relates a database fact syntactically with its D value. Hence, within this framework the DP property is equivalent to the notion of time-fragment preserving, cf. both the definition of an "Interval-based Operator" in [3], and the definition of the SR and SEQ algebras in Section 3 which both by definition are DP.

From the above SR examples wee see that a point-based evaluation of expressions is not the case, i.e., a query is not evaluated for each snapshot in turn comprising a set of (indexed) snapshot results. Even though, this, in fact, could be

the conceptual evaluation model of a combined SR and PB language. However, for a DP language conceptually all snapshots of a database facts is present at each evaluation step where the database fact is involved, i.e., the snapshots are regarded as a collection, i.e., sequence, of value equivalent snapshots.

Further utilizing this knowledge of evaluating over sequences of snapshots, we now go beyond SR (cf. Definition 2.2 where the X refers to conventional parameter expressions) by allowing user-specified dimension expressions in combination with the built-in dimension semantics of a language. This property of a langauge is termed *dimensional parameter expressions* (DPE).

Definition 2.4 An algebra (or query language) which allows parameter expressions X to contain references to D attributes for projections, e.g., $\pi_{A_2,A_3,D}(r)$, and/or dimensional predicates and functions for selections (i.e., restrictions), e.g., $\sigma_{contains(r_1.D,r_2.D)}(r_1 \times r_2)$, supports the *dimensional parameter expression* (DPE) property. ■

For example, envison the above TSQL2 query (similar in STSQL), further restricted to determine employment histories with a duration of more than seven years,

```
TSQL2:  SELECT dept FROM EMP
        WHERE  DURATION(D) > 7;
```

Which yields one tuple, i.e., $\langle d_1, [1988 - 97]\rangle$. So far we have studied several properties of languages, namely SR, PB, DP and DPE. In general the following combinations are possible:

SR & PB	SR & DP	DPE & PB	DPE & DP

The property combinations are incorporated by the algebras to be defined next. That is, the SR algebra combines the SR and DP properties. The SEQ algebra combines the DPE and DP properties. Note, that the underlying algebras of TSQL2 and STSQL are DPE & PB and DPE & DP, respectively. The TSQL2 combination is not considered by the comparison. Moreover, the extrinsic ADT^P and $\mathrm{ADT}^{U/F}$ algebras are in Section 3 classified as a DPE language and a DPE & PB language, respectively. Note that, in general, a PB property excludes a DP property, and that a SR property excludes a DPE property. In Section 4 the comparison uses the above set of properties to expose the differences and similarities in expressive power.

3 The Algebras

This section defines the algebras which are considered by the comparison of the subsequent section.

Pure ADT: The ADT^P algebra differs from the RA operator set by the property of DPE, i.e., by predicate P and the attribute list X expressions, which may involve dimension parameters. However, this is not directly affecting the operator definitions as such:

$$
\begin{aligned}
\pi_X(r) &\triangleq \{t \mid \exists t' \in r\,(t = t'[X])\} \\
\sigma_P(r) &\triangleq \{t \mid t \in r \wedge P(t)\} \\
r_1 \times r_2 &\triangleq \{t_1 \circ t_2 \mid t_1 \in r_1 \wedge t_2 \in r_2\} \\
r_1 \cup r_2 &\triangleq \{t \mid t \in r_1 \vee t \in r_2\} \\
r_1 \setminus r_2 &\triangleq \{t \mid t \in r_1 \wedge t \notin r_2\} \\
\xi_{\langle X,f\rangle}(r) &\triangleq \{t \circ y \mid \exists t' \in r(t = t'[X] \wedge \\
&\qquad y = f(\{t'' | t'' \in r \wedge t''[X] = t'[X]\}))\}
\end{aligned}
$$

An attribute list X in the above definition may

denote a dimension attribute D. A predicate P is on the form $B\theta C$ or $B\theta c$, or several of these expressions combined by logical connectives of $\wedge$ (and), $\vee$ (or) and $\neg$ (not) in the conventional way. Moreover, B and C are attribute names or spatio-temporal function expressions, i.e., $duration(D)$ and $length(D)$, and c is a constant. Finally, $\theta \in \{=, <, >, \leq, \geq, \neq\}$, which is extended with Egenhofer's spatial operator set [6], i.e., $disjoint$, $equals$, $overlaps$, $touches$, in, and $contains$, and Allen's temporal operator set [1], i.e., $before$, $equals$, $overlaps$, $during$, $start$, and end, when B, C and c are of spatial or temporal types, respectively.

Pure ADT with unfold/fold: The $\text{ADT}^{U/F}$ algebra extends the ADT^{P} algebra with the $unfold$ and $fold$ algebraic operators, which allows user-specified simulation of point-based expressions. Pictorially, unfolding is to flatten a relation on its D attribute, i.e., transform the relation into a point-based representation. On the other hand, folding is to "recompute" a more compact representation of a relation where each tuple's D value is the maximal contiguous extent over which a set of A-value equivalent input tuples are defined. See the following example:

A	D
a	[2-3]
a	[3-4]

$\xrightarrow{unfold_D}$

A	D
a	[2-2]
a	[3-3]
a	[4-4]

$\xrightarrow{fold_D}$

A	D
a	[2-4]

The rational behind unfolding is to give a point-based interpretation of the database facts. However, this requires an explicit point-based representation of database facts, to ensure that the operations actually operates on point referenced database objects. Thus, the definition of unfolding replaces each tuple in r with a set of A-value equivalent result tuples, where each result tuple accounts for a distinct point of the D-value associated with the input tuple. More formally,

$$unfold_D(r) \triangleq \{t \mid \exists t' \in r\,(t[A] = t'[A] \wedge t'[D]^s \leq t[D]^s = t[D]^e \leq t'[D]^e)\}$$

Folding enables a more compact representation of the pointwise interpretation of database facts, recall the syntactic correspondence a PB language has to a D-value. The folding operator constructs a single result tuple from two tuples of each set of A-value equivalent input tuples of a relation, where the tuples comprise a maximal chain of adjacent and overlapping D-values. The D-value of the result tuple is, then, denoting this maximal chain. In the definition of $fold_D$ in Figure 2 a), there are three main constraints. First, line one ensures that there exist two A-value equivalent tuples, t_1 and t_2 in r, which contributes to the construction of t with a valid D value. Second, lines two and three ensure that there is a chain of A-value equivalent tuples which comprise a contiguous chain of D values, i.e., for every tuple t_3 in the chain there exists a tuple t_4 which comes "before" in the chain, alternatively $t_3 = t_4$. Note, the "chain" includes at least one tuple, e.g., when $t_1 = t_2$. Finally, lines four and five ensure that this set is maximal, i.e., there does not exist a tuple t_5 that is both A-value equivalent with the tuples in the chain and has a D-value that extends the chain in either of its ends.

There are some important points which need to be clarified, and which differentiate the $\text{ADT}^{U/F}$ approach with respect to other extensions to the relational model and algebra. First, the definition of folding in $\text{ADT}^{U/F}$ is equivalent to a coalesce operator ([11]) of temporal databases, and may be applied to

any relation with a D attribute. Thus, the *fold* operator of $\text{ADT}^{U/F}$ is different to the *fold* operator of the IXSQL algebra [13], which is only applicable to input relations where the D-values are points, e.g., due to a previous flattening of a relation by an IXSQL unfolding. In that sense the $\text{ADT}^{U/F}$ fold operator subsumes the IXSQL fold operator, because $\text{ADT}^{U/F}$ may be applied to arbitrary relations where D-values are not necessarily on the form $[p, p]$. Second, the folding and unfolding of $\text{ADT}^{U/F}$ is both intentionally and semantically different compared with nesting and unnesting as defined for nested or non first normal form (N1NF) relations, e.g., see [10, 16]. Thus, the intension of $\text{ADT}^{U/F}$ is to simulate a pointwise evaluation of expressions, and, in particular, for folding user-coalesce relations over their dimension values. The intension of the N1NF relational model is to be able to manage complex database facts more explicit through an implied hierarchical structure of nested relations, where the nest and unnest operators ([10]) convert back and forth between flattened and nested relations, respectively. The below informal example illustrates the semantic difference between the two approaches (following the fold/unfold example above). We have to assume that the N1NF model support intervals through a system provided ADT:

$$r = \{\langle a, [2-3]\rangle, \langle a, [3-4]\rangle\}$$

$$\begin{aligned} &nest_{D=(D)}(unnest_{D=(D)}(r)) \\ &\quad = nest_{D=(D)}(\{\langle a, [2-3]\rangle, \langle a, [3-4]\rangle\}) \\ &\quad = \{\langle a, \{[2-3], [3-4]\}\rangle\} \end{aligned}$$

First, the unnest operation does not affect the content of r, because intervals are system provided data types. Second, nesting operates on sets, whereas folding operates according to a contiguous relation given by the total order of the elements of an underlying dimension space.

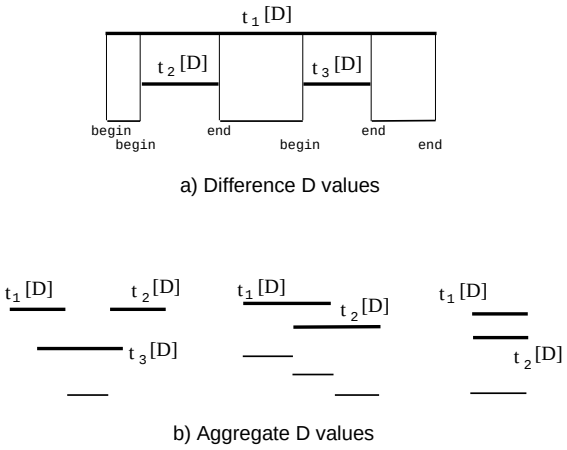

Figure 1: D values of Dimensional Difference and Aggregate Formation

SR: The SR operator set, as defined in Figure 2 b), is explained in the following, where the function signatures, i.e., of the superscripts, indicate the dimension attributes involved by the built-in processing and prevent attribute name conflicts in subsequent operations of an expression. The SR slice operator, i.e., $\tau_p^{sr:D\rightarrow}$, denotes a snapshot database at dimension point p. The projection, selection and union are similar to their RA counterparts, but with the distinction that tuples may contain D values. Note also that due to the SR property no D attribute is allowed referenced in an attribute list X by a projection or an aggregate formation operation, and no dimension predicate or function expressions are allowed by a selection, i.e., the SR algebra does not support the Egenhofer and Allen operator sets.

The Cartesian product combines pairs of candidate tuples of r_1 and r_2 with non-empty intersecting D-values. The D-value of the result tuple is computed as the intersection. Formalizing

$$
\begin{aligned}
fold_D(r) \triangleq\ & \{t \mid \exists t_1 \in r\, \exists t_2 \in r\, (t[A] = t_1[A] = t_2[A] \wedge t[D]^s = t_1[D]^s \wedge t[D]^e = t_2[D]^e \wedge t[D]^s \leq t[D]^e) \wedge \\
& \forall t_3 \in r(t[A] = t_3[A] \wedge t[D]^s \leq t_3[D]^s \leq t[D]^e \Rightarrow \\
& \quad \exists t_4 \in r(t[A] = t_4[A] \wedge (t_4[D]^s \leq t_3[D]^s \leq t_4[D]^e \vee t_3[D]^s = succ(t_4[D]^e)))) \wedge \\
& \neg\exists t_5 \in r\, (t[A] = t_5[A] \wedge (t[D]^s = succ(t_5[D]^e) \vee t_5[D]^s = succ(t[D]^e) \vee \\
& \quad t_5[D]^s < t[D]^s \leq t_5[D]^e \vee t_5[D]^s \leq t[D]^e < t_5[D]^e))\}
\end{aligned}
$$

a) Definition of Folding

$$
\begin{aligned}
\tau_p^{sr:D\rightarrow}(r) &\triangleq \{t \mid \exists t' \in r\, (t = t'[A] \wedge t'[D]^s \leq p \leq t'[D]^e)\} \\
\pi_X^{sr:D\rightarrow D}(r) &\triangleq \{t \mid \exists t' \in r\, (t[X] = t'[X] \wedge t[D] = t'[D])\}, \text{ where } D \text{ does not occur in } X. \\
\sigma_P^{sr:D\rightarrow D}(r) &\triangleq \{t \mid t \in r \wedge P(t)\}, \text{ where } D \text{ does not occur in } P. \\
r_1 \cup^{sr:D_1\times D_2\rightarrow D} r_2 &\triangleq \{t \mid t \in r_1 \vee t \in r_2\} \\
r_1 \times^{sr:D_1\times D_2\rightarrow D} r_2 &\triangleq \{\langle t' \circ t'' \circ d\rangle \mid \exists t_1 \in r_1\, \exists t_2 \in r_2\, (t' = t_1[A] \wedge t'' = t_2[A] \wedge \\
&\qquad d = intersection(t_1[D_1], t_2[D_2]) \wedge \neg disjoint(t_1[D_1], t_2[D_2]))\} \\
r_1 \setminus^{sr:D_1\times D_2\rightarrow D} r_2 &\triangleq \{t \mid \exists t_1 \in r_1\, (t[A] = t_1[A] \wedge \\
&\quad (\exists t_2 \in r_2(t_1[A] = t_2[A] \wedge t_1[D_1]^s \leq t_2[D_2]^e \wedge t[D]^s = succ(t_2[D_2]^e)) \vee \\
&\qquad t[D]^s = t_1[D_1]^s) \wedge \\
&\quad (\exists t_3 \in r_2(t_1[A] = t_3[A] \wedge t_1[D_1]^e \geq t_3[D_2]^s \wedge t_3[D_2]^s = succ(t[D]^e)) \vee \\
&\qquad t[D]^e = t_1[D_1]^e) \wedge \\
&\quad t[D]^s \leq t[D]^e \wedge \\
&\quad \neg\exists t_4 \in r_2(t_1[A] = t_4[A] \wedge \neg disjoint(t[D], t_4[D_2])))\} \\
\xi_{\langle X,f\rangle}^{sr:D\rightarrow D}(r) &\triangleq \{\langle t \circ y \circ d\rangle \mid t_1 \in r \wedge t_2 \in r \wedge t = t_1[X] = t_2[X] \wedge d \in compose(t_1[D], t_2[D]) \wedge \\
&\quad y = f(\{t' \mid t' \in r \wedge t'[X] = t \wedge t'[D]^s \leq d^s \wedge d^e \leq t'[D]^e\}) \wedge \\
&\quad \neg\exists t_3 \in r\, (t_3[X] = t \wedge ((d^s \leq t_3[D]^s \leq d^e \wedge d^e < t_3[D]^e) \wedge \\
&\qquad (d^s \leq t_3[D]^e \leq d^e \wedge t_3[D]^e < d^s)) \wedge \\
&\quad d^s \leq d^e\}, \quad \text{where } D \text{ not in } X, \text{ and } f \text{ belongs to } R \setminus D.
\end{aligned}
$$

b) Definition of the SR Algebra

$$
\begin{aligned}
\pi_X^{seq:D\rightarrow D}(r) &\triangleq \{t \mid \exists t' \in r\, (t[X] = t'[X] \wedge t[D] = t'[D])\} \\
\sigma_P^{seq:D\rightarrow D}(r) &\triangleq \{t \mid t \in r \wedge P(t)\} \\
r_1 \times^{seq:D_1\times D_2\rightarrow D} r_2 &\triangleq \{\langle t_1 \circ t_2 \circ d\rangle \mid t_1 \in r_1 \wedge t_2 \in r_2 \wedge d = intersection(t_1[D_1], t_2[D_2]) \wedge \\
&\qquad \neg disjoint(t_1[D_1], t_2[D_2]))\}
\end{aligned}
$$

c) Definition of the SEQ Algebra

Figure 2: The Folding Operator, and SR and SEQ Algebras

this subspace relationship:

Definition 3.1 Let r_1 and r_2 be two relations. Then, tuples $t_1 \in r_1$ and $t_2 \in r_2$ forms an *intra-subspace* relationship, iff

$$\langle t_1[A] \circ t_2[A] \circ d \rangle \in (r_1 \times^{sr:D_1 \times D_2 \to D} r_2) \quad \blacksquare$$

The difference operator is more involved, and denotes tuples constructed from tuples in r_1, which are referenced by some subspace that are not referenced by any A-value equivalent tuple in r_2. Thus, line one ensures there is a candidate A-valued tuple in r_1. A new D value is computed by lines two through five, where Figure 1 a) illustrates the interesting and intersecting r_2 tuple cases, and indicates the begin and end points which contributes to the computation of a new D value. In the case of Figure 1 a) the difference operator would yield three new tuples. Hence, in general at most three A-value equivalent tuples suffice to compute any resulting D value, i.e., at least one tuple from r_1, and possible one or two tuples from r_2. Lines two and three of the definition in Figure 2 determine the possible $t[D]^s$ points, i.e., given by tuples $t_2 \in r_2$ and $t_1 \in r_1$, respectively. Similar, lines four and five determine the possible $t[D]^e$ points. The $succ$ function is applied to ensure that a result D-value does not intersect with the D-value of the r_2 tuple which contributes to the computation of it. Finally, the last two lines ensure a valid result: Line six ensures that $t[D]$ is valid; Line seven ensures that all r_2 tuples are considered, i.e., there are no A-value equivalent tuples in r_2 intersecting with $t[D]$.

The aggregate formation operator also computes a new D value for each result tuple. From the definition in Figure 2 b), line one ensures that there are two (not necessary distinct) operand tuples that both agree on their grouping attributes, and from which the d-value of the result tuple is composed (see definition of $compose$ below). Line two denotes the aggregate set for which f computes the aggregate value given by y. The characterization of an aggregate set is first that all tuples in the set agree exactly on the same grouping attributes as does the tuples of line one, and second that the D-value associated with each tuple in the aggregate set contains (or equals) the composed d-value. Lines three and four ensure that all candidate tuples are accounted for, i.e., there exists no tuple $t_3 \in r$ which agrees on the grouping attributes and where d and $t_3[D]$ intersect. The last line ensures that the aggregate d value is valid. Notice, that according to Figure 1 b) the two tuples of line one may compose an interval denoting a gap between them. Then, these tuples are not in the aggregate set, but, there is at least one tuple in r with qualified grouping attributes and a D-value that contains this gap, e.g., see $t_3[D]$ of the leftmost example in Figure 1 b). The examples of Figure 1 b) are captured by the compose function given by the following definition:

$$\begin{aligned} &compose(d_1, d_2) \triangleq \{d \mid \\ &\quad (d = d_1 = d_2) \vee (d^s = d_1^e + 1 \wedge d^e = d_2^s - 1) \vee \\ &\quad (d^s = d_1^s \wedge d^e = d_2^s - 1) \vee (d^s = d_1^s \wedge d^e = d_2^e) \vee \\ &\quad (d^s = d_1^e + 1 \wedge d^e = d_2^e) \vee (d^s = d_2^s \wedge d^e = d_1^e)\} \end{aligned}$$

Finally, each SR algebraic operator is snapshot reducible to its RA counterpart according to the definition of Section 2.2, and in terms of the above defined slice operator, e.g., for the selection we have $\tau_p^{sr:D\to}(\sigma_P^{sr:D\to D}(r)) = \sigma_P(\tau_p^{sr:D\to}(r))$.

SEQ: The SEQ algebra goes beyond the SR property by allowing DPE. The redefinitions of three SR operators comprise the transition from

a SR to a SEQ algebra. The definitions of Figure 2 c) show the principle differences by the lack of preconditions for projection and selection operators, and that the Cartesian product explicitly exposes the dimension attributes of its operands. That is, for a result tuple on the form $\langle t_1 \circ t_2 \circ d \rangle$, both the operands tuples $t_1 \in r_1$ and $t_2 \in r_2$ contribute to the result as they are. For example, (leaving out the superscripts) $\sigma_{duration(r.D)<duration(s.D)}(r \times s)$, $\pi_{r.A,s.D}(r \times s)$ and $\pi_D(r)$ are all well-formed SEQ expressions. The schemes associated with the results of these expressions are $(r.A, r.D, s.A, s.D, D)$, $(r.A, s.D, D)$ and ("D", D), respectively, where $r.D$, the two $s.D$, and "D" are only regarded as ordinary ADT attributes. This means that these attributes are exposed. The Cartesian product does built-in exposure, and the two projections do user-specified exposures. The Cartesian product needs to expose attributes to utilize the DPE property. However, a SEQ evaluation discards built-in exposures at certain critical steps. The exposures are managed during an evaluation in terms of an *expose set*, denoted by $e(EXP)$, and the following assignments to the expose set for each step of an evaluation:

$$
\begin{aligned}
e(r) &:= \emptyset \\
e(\pi_X(EXP)) &:= \emptyset \\
e(\sigma_P(EXP)) &:= e(EXP) \\
e(EXP_1 \times EXP_2) &:= e(EXP_1) \cup e(EXP_2) \cup \\
&\quad \{EXP_1.D_1, EXP_2.D_2\} \\
e(EXP_1 \; op \; EXP_2) &:= \emptyset, \text{ where } op \in \{\cup, \setminus\} \\
e(\xi_{\langle X,f \rangle}(EXP)) &:= \emptyset
\end{aligned}
$$

The semantics of an SEQ evaluation, involving the exposures by subexpressions, are given in terms of denotational semantics symbolized by expressions enclosed by $[\![.]\!]$, see below. An expression EXP may involve subexpressions, where an evaluation of a subexpression may involve (implicitly) the above corresponding exposure assignment. However, some subexpressions have to discard SEQ exposed dimension attributes, i.e., not user-exposed dimension attributes, before the subexpression them self are evaluated. This is to ensure that SEQ evaluates naturally and accordingly to the relational schemes assumed by the user. Discarding exposures is required before union compatible operations and upon termination of evaluation. In the first case the presence of exposed attributes may break with user-assumed union-compatible relations. In the second case exposed attributes are generally of no interest beside that they have been input to dimension computation. The discarding is managed by a so-called *complement project* operator[1]: $\overline{\pi}_X(r) \triangleq \pi^{seq:D \to D}_{\{A_1,\ldots,A_n\} \setminus X}(r)$, where X represents the set of previous exposed, and not yet discarded, attributes of an expression EXP that has yielded r. In the below denotations (where E is a shorthand for EXP) it is assumed that discarding of exposed dimension attributes will occur before updating the expose set, e.g., see the denotation of union.

$$
\begin{aligned}
[\![r(R)]\!] &\triangleq r(R) \subseteq dom(A_1) \times \cdots \\
&\qquad \cdots \times dom(A_n) \\
[\![\sigma^{D \to D}_P(E)]\!] &\triangleq \sigma^{seq:D \to D}_P([\![E]\!]) \\
[\![\pi^{D \to D}_X(E)]\!] &\triangleq \pi^{seq:D \to D}_X([\![E]\!]) \\
[\![\xi^{D \to D}_{\langle X,f \rangle}(E)]\!] &\triangleq \xi^{seq:\{D\} \to D}_{\langle X,f \rangle}([\![(E)]\!]) \\
[\![E_1 \times^{D_1 \times D_2 \to D} E_2]\!] &\triangleq [\![E_1]\!] \\
&\qquad \times^{seq:D_1 \times D_2 \to D} \\
&\qquad [\![E_2]\!] \\
[\![E_1 \cup^{D_1 \times D_2 \to D} E_2]\!] &\triangleq \overline{\pi}_{e(E_1)}([\![E_1]\!]) \\
&\qquad \cup^{seq:D_1 \times D_2 \to D} \\
&\qquad \overline{\pi}_{e(E_2)}([\![E_2]\!]) \\
[\![E_1 \setminus^{D_1 \times D_2 \to D} E_2]\!] &\triangleq \overline{\pi}_{e(E_1)}([\![E_1]\!]) \\
&\qquad \setminus^{seq:D_1 \times D_2 \to D} \\
&\qquad \overline{\pi}_{e(E_2)}([\![E_2]\!]) \\
[\![\bot \, E \, \top]\!] &\triangleq \overline{\pi}_{e(E)}([\![E]\!])
\end{aligned}
$$

Summary of Properties: In this section we also defined the notion of intra-subspace relationships to characterize the class of relations denoted by the Cartesian product of SR, and which generalizes directly to SEQ, and gener-

[1]The term complement is used because the projection list is "complementary" to the list of a regular projection.

alizes to ADT^P and ADT$^{U/F}$ with an addition of a $\neg disjoint$ parameter expression. However, ADT^P and ADT$^{U/F}$ also support a complementary class of non-empty relations:

Definition 3.2 Let r_1 and r_2 be relations. Then, $t_1 \in r_1$ and $t_2 \in r_2$ forms an *inter-subspace* relationship, iff,

$$\langle t_1 \circ t_2 \rangle \in \sigma_{disjoint(t_1[D_1],t_2[D_2])}(r_1 \times r_2) \blacksquare$$

We now state the following lemmas to further formalize the characteristics of the algebras:

Lemma 3.3 The SR and SEQ algebras do not denote the class of relations with inter-subspace relations as defined by Definition 3.2.

Proof: Since the Cartesian product is fundamental, only this operator could be used to combine tuples of distinct relations. However, in SR and SEQ this operator combines, by definition, only by intra-subspace relationship, cf. Definition 3.1. ■

Lemma 3.4 An algebra that by definition is SR is not DPE.

Proof: Follows directly from the definitions of SR and DPE, cf. Section 2.2. ■

Lemma 3.5 An algebra that by definition is PB is not DP.

Proof: This follows directly from the definitions of PB (Definition 2.3 and DP (cf. [3]). See also the TSQL2 and STSQL examples of Section 2.2. ■

The following table summarizes the dimensional characteristics of each of the algebras defined in this section.

ADT^P	**ADT$^{U/F}$**	**SR**	**SEQ**
inter-rel.	inter-rel.		
intra-rel.	intra-rel.	intra-rel.	intra-rel.
DPE	DPE & PB	SR & DP	DPE & DP

4 Comparison of Expressive Power

The comparison of equivalent expressions by the notion of *strict equivalence* (i.e., Section 4.1 below), is based on the following structure: Let Q_1 and Q_2 be two (algebraic) languages, then, $Q_1 \leq Q_2$ means that Q_1 is at most up to equal expressive with respect to Q_2 if $\forall\, q_1 \in Q_1\, \exists\, q_2 \in Q_2\, (q_1 \equiv q_2)$. Moreover, we must assume that these queries are expressed over arbitrary data structures, i.e., any relation extended with a dimensional ADT in our case. Then, the $q_1 \equiv q_2$ above is equivalent to:

$$\forall\, db([\![q_1(db)]\!]_{Q_1} = [\![q_2(db)]\!]_{Q_2}),$$

where db is a data structure, and $[\![q_i(db)]\!]_{Q_i}$, $1 \leq i \leq 2$, is the result of evaluating q_i over a database db according to the semantics of language Q_i. (Subscript Q_i is in the following given by the context, and omitted). Moreover, from the above we deduce $Q_1 < Q_2 \stackrel{\Delta}{\Longleftrightarrow} Q_1 \leq Q_2 \wedge \neg(Q_2 \leq Q_1)$, i.e., language Q_1 is subsumed by language Q_2.

4.1 Comparison by Strict Equivalence

ADT^P vs. ADT$^{U/F}$ The RA framework, as defined in Section 2, does not allow any user-specified constructions of attributes values, so the following theorem summarizes the corre-

spondence between ADT^P and $\mathrm{ADT}^{U/F}$. Notice that we regard an interval as a single ("atomic") value, and not as two explicit begin and end RA attributes. Isolated to intervals this approach could be argued. However, when more complex and irregular spatial attribute values are involved, such as polylines and polygons, this approach reflects the ADT extensions to RA in general.

Theorem 4.1 $\mathrm{ADT}^P < \mathrm{ADT}^{U/F}$

Proof: According to the comparison structure the proof is on the following form:

$$(\forall\, q_1 \in \mathrm{ADT}^P\, (\exists\, q_2 \in \mathrm{ADT}^{U/F}(q_1 \equiv q_2)))\wedge$$
$$(\exists\, q_3 \in \mathrm{ADT}^{U/F}\, (\forall\, q_4 \in \mathrm{ADT}^P(\neg(q_3 \equiv q_4))))$$

The lhs (left hand side) of the conjunction is given directly by the definitions of ADT^P and $\mathrm{ADT}^{U/F}$ in Section 3. That is, every ADT^P expression is also an $\mathrm{ADT}^{U/F}$ expression. Put differently, $\mathrm{ADT}^{U/F}$ is defined in terms of the operator set of ADT^P plus the *unfold* and *fold* operators.

For the rhs (right hand side), envision the dimensional aggregate formation query of Section 2.2:

1) Determine the (periodic) salary pay-outs for each department over all times,

given by the equivalent $\mathrm{ADT}^{U/F}$ expression:

$$\mathit{fold}_D(\xi_{\langle\{\mathtt{dept},D\},sum_3\rangle}(\mathit{unfold}_D(\mathtt{EMP}))),$$

which, in fact, yields the same result as depicted by `RESULT of 1)` of Section 2.2. Each aggregate set of this expression is denoted by the set of tuples that mutually agree on both their `dept` and D values, where D-values are on the form $[p,p]$, due to $\mathit{unfold}_D(\mathtt{EMP})$. Moreover, by definition an aggregate set yields a result tuple t on the form $\langle t' \circ y\rangle$, where $t' = t[\{\mathtt{dept}, D\}]$ and $y = sum_3$. This is a point-based aggregation over the relation `EMP`, and the subsequent folding coalesces each set of result tuples, where tuples both denote the same aggregate and comprise a maximal contiguous chain by their respective D-values. The ADT^P algebra is not capable of simulating this fragmentation into point referenced database facts, i.e., otherwise *unfold* and *fold* would not have been fundamental point-based operators within this framework. This finalize the proof of showing that $\mathrm{ADT}^{U/F}$ subsumes ADT^P. ■

Note that the subset of expressions in $\mathrm{ADT}^{U/F}$ which involves *unfold* or *fold* operators, or both, is, in general, not corresponding to any subset of expressions in ADT^P. This means that where RA is only extended with abstract data types, as in the case of the ADT^P algebra, a dimensional interpretation of database facts is not an underlying property of such an algebra. In particular, the ADT^P algebra does not express the class of coalesced queries, and not the class of dimensional queries, e.g., the dimensional aggregation formation as presented above.

SR vs. SEQ Based on the SR and SEQ definitions of Section 3, respectively, the following theorem states that SR is subsumed by SEQ.

Theorem 4.2 SR < SEQ

Proof: Analogously to Theorem 4.1 the proof is by showing:

$$(\forall\, q_1 \in SR\, (\exists\, q_2 \in SEQ(q_1 \equiv q_2)))\wedge$$
$$(\exists\, q_3 \in SEQ\, (\forall\, q_4 \in SR(\neg(q_3 \equiv q_4))))$$

The lhs (left hand side) of the conjunction is for the slice, union, difference and aggregation formation operators directly given by identical definitions in both SR and SEQ, cf. the definitions of

the algebras in Section 3. Moreover, the SR projection and SR selection are only more restrictive than the respective SEQ operators due to the property of DPE of SEQ .

Due to dimension attribute exposures by SEQ, we have to prove that the SR Cartesian product is equivalent with the SEQ Cartesian product proved according to our comparison structure of strict equivalence:

$$\forall\, r_1, r_2\, ((r_1 \times^{sr:D_1\times D_2\to D} r_2) = \overline{\pi}_{e(r_1 \times^{D_1\times D_2\to D} r_2)}(r_1 \times^{seq:D_1\times D_2\to D} r_2))$$

Let t be in the lhs of the equality, then, by the definition of the SR Cartesian product there exists tuples t_1 in r_1 and t_2 in r_2, such that $t[r_1.A] = t_1[A]$, $t[r_2.A] = t_2[A]$ and $t[D] = intersection(t_1[D], t_2[D])$. Since, $t_1 \in r_1$ and $t_2 \in r_2$, there is according to the definition of the SEQ Cartesian product a tuple $t' \in (r_1 \times^{seq:D_1\times D_2\to D} r_2)$, such that $t'[r_1.A] = t_1[A]$, $t'[r_2.A] = t[A]$, $t'[D] = intersection(t_1[D], t_2[D])$, $t'[r_1.D_1] = t_1[D_1]$, and $t'[r_2.D_2] = t_2[D_2]$. Then, by applying the complement projection, according to the evaluation by denotation as defined for the SEQ algebra in Section 3, we get (recall that the complement projection could be "rewritten" into a projection):

$$\overline{\pi}_{e(r_1 \times^{D_1\times D_2\to D} r_2)}(t') = \pi^{seq:D\to D}_{r_1.A, r_2.A}(t') = t$$

Thus, every tuple in the SR Cartesian product is also in the SEQ Cartesian product. The opposite inclusion is given by the same strategy and omitted.

The rhs (right hand side) of the conjunction is given by Lemma 3.4, and exemplified by a SEQ version of the query of Section 2.2, which determines employment histories of departments with durations of more than seven years:

$$\pi^{seq:D\to D}_{\texttt{dept}}(\sigma^{seq:D\to D}_{duration(D)>7}(\texttt{EMP})) = \{\langle d_1, [1988-97]\rangle\}$$

Notice the user-specified reference to the dimension attribute. Thus, SR is subsumed by SEQ. ■

ADT$^{U/F}$ vs. SEQ The following theorem states the correspondence of ADT$^{U/F}$ and SEQ algebras.

Theorem 4.3 Neither SEQ $\leq$ ADT$^{U/F}$ nor ADT$^{U/F} \leq$ SEQ.

Proof: The proof is on the following from:

1)$\exists q_1 \in SEQ(\forall q_2 \in \text{ADT}^{U/F}(\neg(q_1 \equiv q_2)))$

2)$\exists q_3 \in \text{ADT}^{U/F}(\forall q_4 \in SEQ(\neg(q_3 \equiv q_4)))$

Claim 1) above is given by the definition of DP (cf. [3]) and Lemma 3.5. Thus, SEQ evaluates expressions where the dimension D-values are preserved by the result. The ADT$^{U/F}$, on the other hand, uses the *unfold* and *fold* operators to simulate a pointwise interpretation of database facts. By Theorem 4.1 we know that only dimensional queries are of interest here. Both the dimensional Cartesian product, dimensional difference and dimensional aggregate formation operations are in the ADT$^{U/F}$ algebra forced to include an *unfold* (and, eventually, *fold*) to obtain the dimensional semantics wanted. However, by the definition of *unfold* the information about the original D values is lost, and a subsequent *fold* operation is not able to restore it completely. The following example illustrate this point: Let relations $r_1 = \{\langle a, [2-3]\rangle, \langle a, [4-6]\rangle\}$ and $r_2 = \{\langle a, [5-6]\rangle\}$. Then, the respective difference operations yield the following results:

$$r_1 \setminus^{seq:D_1\times D_2\to D} r_2 = \{\langle a, [2-3]\rangle, \langle a, [4-4]\rangle\},$$
$$fold_D(unfold_D(r_1) \setminus unfold_D(r_1)) = \{\langle a, [2-4]\rangle\}.$$

Claim 2) is showed by Lemma 3.3. Since ADT$^{U/F}$ subsumes ADT^P by Theorem 4.1, let the following ADT^P expression be issued on the sample database of Section 2.2, which illustrates the non-correspondence with any SEQ expressions:

$$\sigma_{disjoint(\texttt{EMP}.D, \texttt{DEPT}.D)}(\texttt{EMP} \times \texttt{DEPT})$$

yields:

$$\{\langle \text{Pete}, d_1, 10k, [1985-90], \text{Ann}, d_1, [1991-95]\rangle\}$$

Then, according to the inter-subspace relationship class of expression supported by ADT^P, and not SEQ, cf. Lemma 3.3 of Section 3, the above query makes no sense in SEQ. Thus, the theorem holds. ■

The result of Theorem 4.3 shows that $\text{ADT}^{U/F}$ is not applicable to query classes inducing DP semantics, whereas SEQ (and SR) by definition is DP. Moreover, inter-subspace relationships are not expressible by SEQ (or SR). Thus, the following is a corollary of Theorem 4.3:

Corollary 4.4 Neither SR $\leq$ ADT^P nor $\text{ADT}^P \leq$ SR.

Proof: The proof follows from Theorem 4.1, Theorem 4.2, and Theorem 4.3. ■

4.2 Comparison by Snapshot Equivalence

Now, recall the SE property (cf. Definition 2.1), which ensures that snapshot relations are equal. The following definition generalizes the SE property to account for expressions of two languages Q_1 and Q_2:

Definition 4.5 Let $q_1 \in Q_1$ and $q_2 \in Q_2$ be two expressions, then, q_1 and q_2 are *snapshot equivalent expressions*, denoted $q_1 \overset{se}{\equiv} q_2$, if for all points p, data structures db, such that

$$\tau_p([\![q_1(db)]\!]) = \tau_p([\![q_2(db)]\!])$$ ■

The following theorem shows that every SEQ operator is snapshot equivalent with a corresponding expression in $\text{ADT}^{U/F}$.

Theorem 4.6 The SEQ operator set is snapshot equivalent ($\overset{se}{\equiv}$ of Definition 4.5) with respect to corresponding expressions in $\text{ADT}^{U/F}$, given by:

$$\begin{aligned}
\tau_p^{seq:D\rightarrow}(r) &\overset{se}{\equiv} \pi_{R\setminus D}(\\
&\quad \sigma_{contains(D,[p,p])\vee}\\
&\quad\ equals(D,[p,p])}(r))\\
\pi_X^{seq:D\rightarrow D}(r) &\overset{se}{\equiv} \pi_{X,D}(r)\\
\sigma_P^{seq:D\rightarrow D}(r) &\overset{se}{\equiv} \sigma_P(r)\\
r_1 \cup^{seq:D_1\times D_2\rightarrow D} r_2 &\overset{se}{\equiv} r_1 \cup r_2\\
r_1 \times^{sr:D_1\times D_2\rightarrow D} r_2 &\overset{se}{\equiv} \mathit{fold}_{r_1.D}(\\
&\quad \pi_{r_1.A,r_2.A,r_1.D}(\\
&\quad\ \sigma_{equals(r_1.D,r_2.D)}(\\
&\quad\quad \mathit{unfold}_D(r_1) \times\\
&\quad\quad\ \mathit{unfold}_D(r_2))))\\
r_1 \setminus^{seq:D_1\times D_2\rightarrow D} r_2 &\overset{se}{\equiv} \mathit{fold}_D(\\
&\quad \mathit{unfold}_D(r_1) \setminus\\
&\quad\ \mathit{unfold}_D(r_2))\\
\xi_{\langle X,f\rangle}^{seq:\{D\}\rightarrow D}(r) &\overset{se}{\equiv} \mathit{fold}_D(\\
&\quad \xi_{\langle\{X,D\},f\rangle}(\mathit{unfold}_D(r)))
\end{aligned}$$

Proof: The proof is by showing each of the above $\overset{se}{\equiv}$, in turn. That is, showing that results are according to $\overset{se}{\equiv}$ of Definition 2.1. The first four equivalences are given directly from the SEQ algebraic definitions in Section 3 and the above corresponding $\text{ADT}^{U/F}$ expressions. The remaining cases are more involved, and their proofs are by inclusion both ways, i.e., showing that an arbitrary lhs (i.e., left hand side) tuple is also in the rhs (i.e., right hand side) result, and

vice versa, with respect to the SE property.

Note that for the SEQ Cartesian product we use the SR Cartesian product, since they are shown equivalent by Theorem 4.2, and we may leave out the discarding of exposure here. Let t be a lhs tuple. Then, by the definition of the SR Cartesian product there exists tuples $t_1 \in r_1$ and $t_2 \in r_1$, such that $t[r_1.A] = t_1[A]$, $t[r_2.A] = t_2[A]$, and $t[D] = intersection(t_1[D], t_2[D])$. By the definition of unfolding there are sets of A-value equivalent tuples, comprising maximal chains of D-values, i.e., $\{t'_1, \ldots, t'_n\} \subseteq unfold(r_1)$ and $\{t''_1, \ldots, t''_m\} \subseteq unfold(r_2)$, so that $t_1[A] = t'_i[A]$, $1 \leq i \leq n$, $t'_1[D]^s = t'_1[D]^e \leq t_1[D]^s \leq t_1[D]^e \leq t'_n[D]^s = t'_n[D]^e$, $t_2[A] = t''_j[A]$, $1 \leq j \leq m$, and $t''_1[D]^s = t'_1[D]^e \leq t_2[D]^s \leq t_2[D]^e \leq t''_m[D]^s = t'_m[D]^e$. This implies that there are tuples from these two sets which are combined by the rhs $\text{ADT}^{U/F}$ Cartesian product, restricted by the rhs $\text{ADT}^{U/F}$ selection, where the predicate simulates the actual dimension intersection computation of the SR Cartesian product, and, finally, projected by the rhs $\text{ADT}^{U/F}$ projection. The result before folding is a set of tuples $\{t'''_1, \ldots, t'''_h\}$, so that $t'''_i[r_1.A] = t_1[A]$, $t'''_i[r_2.A] = t_2[A]$, thus, $t'''_i[A] = t[A]$, $1 \leq i \leq h$, and $t'''_1[r_1.D]^s = t'''_1[r_1.D]^e \leq t[D]^s \leq t[D]^e \leq t'''_h[r_1.D]^s = t'''_h[r_1.D]^e$. By the definition of folding, the coalesced result tuple, i.e., $fold_{r_1.D}(\{t'''_1, \ldots, t'''_h\}) = t_3$, has a $t_3[r_1.D]$-value which either equals or contains the lhs $t[D]$. By $\overset{se}{\equiv}$ the inclusion holds. The opposite direction is similar and details are omitted. However, briefly note that for each rhs tuple t', which results from the rhs Cartesian product, selection, projection, and, finally, coalesced by $fold_D$, there are two sets of A-value equivalent tuples $S_1 \subseteq unfold_D(r_1)$ and a subset $S_2 \subseteq unfold_D(r_2)$, where each such set denotes a maximal contiguous chain (i.e., a connected set of points), say c_1 and c_2, respectively, comprised by the D-values associated with the tuples. Hence, $t'[D]$ corresponds to the non-empty intersection of c_1 and c_2. Moreover, S_1 and S_2 must correspond to $S'_1 = \{t_1, ..., t_n\} \subseteq r_1$, $1 \leq n$, and $S'_2 = \{t'_1, ..., t'_m\} \subseteq r_2$, $1 \leq m$, respectively. Both S'_1 and S'_2 denote a maximal chain of overlapping or adjacent D-values in r_1 and r_2, respectively. By definition the SEQ Cartesian product, i.e., by combining S'_1 and S'_2, yields a lhs result set $\{t''_1, ...t''_k\}$, $1 \leq k \leq nm$. This implies that, the rhs tuple t' is A-value equivalent with every lhs t''_i, $1 \leq i \leq k$. It also implies that either $t'[D] = t''_i[D]$, for k=1 (and the inclusion follows directly), or $t'[D]$ contains every $t''_i[D]$, for $k > 1$. That is, the set of $t''_i[D]$-values, is the non-unfolded version of the intersection of the c_1 and c_2 chains, an intersection which equals $t'[D]$. Thus, by $\overset{se}{\equiv}$ the inclusion holds.

For the dimensional difference, let t be a tuple of the rhs. By the definition of folding there exists a set of tuples $\{t_1, \ldots, t_n\} \subseteq (unfold_D(r_1) \setminus unfold_D(r_2))$, so that $t[A] = t_i[A]$, $t[D]^s \leq t_i[D]^s = t_i[D]^e \leq t[D]^s$, $1 \leq i \leq n$. There are two options: The first option is that there exists an identical subset in $unfold_D(r_1)$, which comprise a maximal chain over the associated D-values. This implies that for all $t''' \in unfold_D(r_2)$, $t''' \notin \{t_1, \ldots, t_n\}$. Moreover, there is a set of one or more A-value equivalent tuples in r_1, from which the unfolded set is constructed. In that case the set of r_1 tuples, due to the DP property of the SEQ difference, contributes as it is to the lhs result. By the definition of $\overset{se}{\equiv}$ the inclusion holds.

The other option is more involved, where $\{t_1, \ldots, t_n\}$ above is a subset of an other set S of A-value equivalent tuples in $unfold(r_1)$,

where S comprises a maximal D-valued chain in $unfold(r_1)$. Then, there exist corresponding A-value equivalent sets $S_i \subseteq unfold(r_2)$, $1 \leq i$, where each S_i both comprises a maximal D-valued chain in $unfold(r_2)$, and has a non-empty intersection with S. Thus, $\{t_1, \ldots, t_n\} \subseteq (S \setminus (\cup_{S_i \subseteq unfold(r_2)}(S_i)))$, where each S_i has the above given properties. By the definition of unfold we get that there exist $S' = \{t'_1, ..., t'_m\} \subseteq r_1$, $1 \leq m$, which corresponds to S, and $S'' = \{t''_1, ..., t''_k\} \subseteq r_2$, $1 \leq k$, which corresponds to $\cup_{S_i \subseteq unfold(r_2)}(S_i)$. Now according to the definition of the SEQ difference the D-value of a lhs tuple t' is constructed, in this case, from either tuple t''' in S' and one tuple in S'', or from two tuples in S''. The tuples in S'' satisfy the intersection condition with a tuple of S'. For example, see the illustration of Figure 1 a). Since, tuples in S' is given so that they comprise a chain of adjacent and overlapping D-values, the SEQ difference ensures that every part of that chain which is not referenced by any tuple in S'' contributes to the result (recall that rhs t is constructed from exactly the same sets of tuples). Moreover, the DP property of SEQ ensures that each such part are given by fragments which preserve the dimension values of the original tuples in S'. That is, SEQ yields a result set $\{t'''_1, \ldots t'''_l\}$, $1 \leq l$, for every qualified part of the chain in S'. This implies that exactly one such result set must correspond to the rhs tuple t, i.e., $fold_D(\{t'''_1, \ldots t'''_l\}) = t$. The rhs PB property folds such a part of a chain into the single tuple t. By $\overset{se}{\equiv}$ the inclusion holds. The opposite direction is analogous to the same direction for the SEQ Cartesian product above and omitted.

Finally, for the dimensional aggregate formation a lhs tuple t is on the form $\langle t'[X] \circ y \circ d \rangle$. By the definition of aggregate formation t corresponds to a non-empty aggregate set of r tuples, say denoted by $\mathcal{X}_d$, where $t' \in \mathcal{X}_d$. The tuples of $\mathcal{X}_d$ agree on their X-values, and have D-values that contain (or equal) the d value associated with t. From $\mathcal{X}_d$ the aggregate function f, computes the y of t. Now, since $\mathcal{X}_d \subseteq r$, then, $\mathcal{X}_d$ contributes to the rhs $unfold_D(r)$ as many times as there are distinct points over the set of $t'[D]$ values, from all $t' \in \mathcal{X}_d$. Say $\{t'_1, ..., t'_n\} \subseteq unfold_D(r)$, $1 \leq n$. The rhs aggregate formation partitions the relation $unfold_D(r)$ into aggregate sets, where any two of the above t'_i and t'_j, for $i \neq j$, $1 \leq i \leq n$, and $1 \leq j \leq n$, are member of distinct aggregate sets of the partition. Note that each $t'_i[D] = [p_i, p_i]$, for $1 \leq i \leq n$, and all p_i's form a connected set of points. Thus, there are n (and only n) aggregate sets, $\mathcal{X}_{[p_1,p_1]}, ..., \mathcal{X}_{[p_n,p_n]}$, all subsets of $unfold_D(r)$, which relate back to the above lhs $\mathcal{X}_d$. However, by the definition of the lhs aggregate formation and the DP property, only a subset of the rhs aggregate sets corresponds directly to the $\mathcal{X}_d$, i.e., every $\mathcal{X}_{[p_i,p_i]}$, where p_i is contained by d. Each such $\mathcal{X}_{[p_i,p_i]}$ must be constructed from exactly the same set of tuples $t' \in r$ as denoted by the set $\mathcal{X}_d$. Let $|d|$ denote the number of points p in the lhs d. Then, the subset of rhs aggregate sets, corresponding to $\mathcal{X}_d$, yields before folding a result set $\{t''_1, \ldots, t''_{|d|}\}$, where tuples are on the form $\langle \langle t'[X], [p_{i+j}, p_{i+j}] \rangle \circ y' \rangle$, $1 \leq i$, $1 \leq (i + j) \leq n$, and $0 \leq j < |d|$. This set preserves exactly the same correspondence with $\mathcal{X}_d$ as did the rhs aggregate sets. Since, the rhs aggregate formation is computed over the corresponding set of tuples as in the lhs aggregation formation, it implies that $y' = y$. Finally, by folding all the rhs aggregate tuples, including $\{t''_1, \ldots, t''_n\}$, we obtain a rhs tuple t'' on the form $\langle \langle t'[X], d' \rangle \circ y \rangle$, where d' either equals or contains d, due to the coalescing of the rhs results. Thus, despite different orderings of lhs and

rhs attributes, we conclude that by $\stackrel{se}{\equiv}$ the inclusion holds. The opposite direction is omitted, but it is similar to the previous equivalences in the sense that one rhs tuple may yield one or more corresponding lhs tuples, and by $\stackrel{se}{\equiv}$ the inclusion holds.

Thus, all the above set of equivalences hold.■

A final result with respect to snapshot equivalence states that SEQ is subsumed by $\mathrm{ADT}^{U/F}$, denoted by $SEQ <_{se} \mathrm{ADT}^{U/F}$. Hence, a similar comparison structure as for strict equivalence is applied:

Theorem 4.7 $SEQ <_{se} \mathrm{ADT}^{U/F}$

Proof: The proof is on the following form:

$$\forall q_1 \in SEQ\,(\exists q_2 \in \mathrm{ADT}^{U/F}(q_1 \stackrel{se}{\equiv} q_2)) \wedge \exists q_3 \in \mathrm{ADT}^{U/F}\,(\forall q_4 \in SEQ(\neg(q_3 \stackrel{se}{\equiv} q_4)))$$

The lhs of the conjunction is given directly by the result of Theorem 4.6.

The rhs of the conjunction follows directly from the inter-subspace example of claim 2) of Theorem 4.3. ■

4.3 Extensions to the Comparison Framework

The motivation behind the above comparison was to investigate and compare dimensional query languages with respect to a certain set of properties and equivalence cirteria. The RA-based framework was defined as simple as possible to emphasize some establised properties of query langauges, herein described by SR, PB, DPE and DP. One biproduct of this comparison is hopefully that the framework, properties and comparison criteria may be generalized to account for more extensible dimensional query languages, i.e., other features that have both theoretical and practical interest. In particular, other features for dimensional query languages may include support of multiple ADT dimensions, dimension function expressions in the projection list to compensate for missing intrinsic D-value computations, set-valued attributes, ADT for "irregular" values (e.g., polygons), etc. We briefly look into both the issue of irregular ADT values, possibly combined with other features, and the issue of multiple ADT dimensions support.

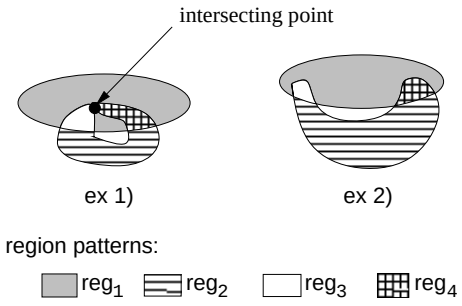

Figure 3: Two examples of intersecting spatial regions

Most spatial query languages are basically non-intrinsic ADT extensions (e.g., cf. [9]). So, let a spatial relation be given by $r((A, D)) = \{\langle a_1, reg_1\rangle, \langle a_2, reg_2\rangle\}$, where the dimension ADT attribute D is of type polygon and denotes simple regions, here exemplified by reg_1 and reg_2. Assume now that we want to determine all a_1-valued tuples in r and their region intersections with non-a_1 valued tuples also in r. Thus, the regions of interest is given pictorially by two examples in Figure 3, where regions reg_3 and reg_4 denote the intersections of both examples. One interesting problem is to study similarities and differences in properties illustrated by a result on the form $\{\langle a_1, reg_3\rangle, \langle a_1, reg_4\rangle\}$ versus a result on the form $\{\langle a_1, \{reg_3, reg_4\}\rangle\}$, i.e., within a framwork where DP and PB are gen-

eralized to characterize spatial semantics of this kind. Moreover, should reg_3 and reg_4 be regarded as one region or separate regions due to the touch relationships (i.e., intersecting boundaries) of ex 1) in Figure 3, etc..

Orthogonal multiple ADT dimensions are an interesting feature, and is established in temporal databases by valid and transaction time dimensions, where orthogonality is given by well-defined sematics, e.g., [11, 22]. However, for extrinsic languages the problem of "indeterminism" arise, due to user-specified simulated dimension evaluation. The following example illustrates this point, where two folding operations are applied to a relation $r(R)$, where $R = (A, D_1, D_2)$, i.e., with two ADT dimensions:

$$\begin{aligned} r(R) &= \{\langle a, [2-2], [4-4]\rangle, \langle a, [1-1], [4-4]\rangle \\ &\quad \langle a, [2-2], [5-5]\rangle\}, \end{aligned}$$

$$\begin{aligned} fold_{D_1}(fold_{D_2}(r)) &= \{\langle a, [2-2], [4-5]\rangle, \\ &\quad \langle a, [1-1], [4-4]\rangle\} \end{aligned}$$

$$\begin{aligned} fold_{D_2}(fold_{D_1}(r)) &= \{\langle a, [1-2], [4-4]\rangle, \\ &\quad \langle a, [2-2], [5-5]\rangle\} \end{aligned}$$

For intrinsic languages the problem is resolved by the well-defined algebra which accounts for multiple orthogonal ADT dimensions.

5 Conclusion

By strict equivalence the $\mathrm{ADT}^{U/F}$ and SEQ algebras differ in two major respects. First, in general, SEQ expressions provide more precise information about the underlying database facts, i.e., by the DP property. Second, the $\mathrm{ADT}^{U/F}$ denotes an inter-subspace class of expressions not supported by the SEQ algebra. By SE, which implies that we relax on the DP property, we showed that the SEQ operators are equivalent with corresponding $\mathrm{ADT}^{U/F}$ expressions. In fact, the $\mathrm{ADT}^{U/F}$ algebra subsumes the SEQ algebra with respect to SE.

The $\mathrm{ADT}^{U/F}$ and SEQ algebras have distinct individual strength, and a user-oriented query language should benefit both from both extrinsic and intrinsic semantics, e.g., cf. STSQL [2]. The framework seems to be a sound basis with respect to both the properties and the comparison criteria defined.

Another observation concerns future extensions of the RA framework. An interesting issue is to investigate to what degree these properties scale and generalize for new extensions. More specific tasks may involve studying extensions in isolation, e.g., "indeterminism" of extrinsic languages, as illustrated in Section 4.3. Finally, with the above findings in mind, the complementary comparison of extrinsic vs. intrinsic languages with respect to convenience of expressions (e.g., the length of "similar" expressions in number of operators, etc.) will further explore differences and similarities of these two approaches.

Acknowledgments: This research has in part been founded by The Research Council of Norway, through grants MOI.31297 (BEST) and 117644/223 (DynaMap).

We thank Christian S. Jensen and Mike Böhlen, both at Ålborg University, for fruitful discussions on both content and structure on earlier drafts of this paper.

References

[1] J .F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[2] M. Böhlen, C. S. Jensen, and B. Skjellaug. Spatio-Temporal DataBase Support for Legacy Applications. In *Proceedings of the ACM Symposium on Applied Computing*, pages 226–234, February/March 1998.

[3] M. H. Böhlen, R. Busato, and C. S. Jensen. Point- Versus Interval-based Temporal Data Models. In *Proceedings of IEEE International Confenrence on Data Engineering*, pages 192–200, February 1998.

[4] J. Chomicki. Temporal Query Languages: A Survey. In Ohlbach H. J. Gabbay, D. M., editor, *Proceedings of the First International Conference on Temporal Logic*, pages 506–534. Lecture Notes in Artificial Intelligence 827, Springer-Verlag, July 1994.

[5] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970.

[6] M. J. Egenhofer and R. D. Franzosa. Point-set topological spatial relations. *International Journal on Geographical Information systems*, 5(2):161–174, 1991.

[7] M. Erwig, M. Schneider, and R.H. Güting. Temporal objects for spatio-temporal models and a comparison of their representations. In *Proceedings of ER'98 Workshop on Spatio-temporal Data Management*, number 1552 in Lecture Notes in Computer Science, pages 454–465. Springer-Verlag, 1998.

[8] O. Etzion, S. Jajodia, and S. Sripada (eds.). *Temporal Databases: Research and Practice*. Springer Verlag, 1998.

[9] R. H. Güting. An Introduction to Spatial Database Systems. *The VLDB Journal*, 3:357–399, 1994.

[10] G. Jaeschke and H. J. Schek. Remarks on the algebra of non first normal form relations. In *Proceedings of ACM Symposium on Principles of Database Systems*, 1982.

[11] C. S. Jensen, J. Clifford, R. Elmasri, S. K. Gadia, P. Hayes, and S. Jajodia [eds]. A Glossary of Temporal Database Concepts. *ACM SIGMOD Records*, 23(1):52–64, March 1994.

[12] Anthony Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the ACM*, 29(3):699–717, July 1982.

[13] N. A. Lorentzos and Y. G. Mitsopoulos. SQL Extension for Interval Data. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):480–499, 1997.

[14] E. McKenzie and R. Snodgrass. An Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Surveys*, 23(4):501–543, December 1991.

[15] J. Melton and A.R. Simon. *Understanding the New SQL: A Complete Guide*. San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1993.

[16] G. Özsoyoğlu, Z. M. Özsoyoğlu, and V. Matos. Extending relational algebra and relational calculus with set-valued attributes and aggregate functions. *ACM Transactions on Database Systems*, 12(4):566–592, December 1987.

[17] J. Paredaens and B. Kuijpers. Data models and query languages for spatial databases. *Data & Knowledge Engineering, Elsevier Science B.V.*, 25:29–53, 1998.

[18] M. Schneider. *Spatial data types for database systems : finite resolution geometry for geographic information systems*, volume 1288 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[19] B. Skjellaug. Temporal Data: Time and Relational Databases. Research Report 246, Department of Informatics, University of Oslo, April 1997. ISBN 82-7368-161-0.

[20] R. T. Snodgrass. The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[21] R. T. Snodgrass (editor). *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers, 1995.

[22] A. U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors. *Temporal Databases: Theory, Design, and Implementation.* Benjamin/Cummings, 1993.