Note that $\mathcal{I}$ includes intervals of length 1. Also let $\wp(\mathcal{I})$ be the power set of $\mathcal{I}$. While $\mathcal{I} \subset \wp(\mathcal{T})$, each element of $\wp(\mathcal{I})$ is a set, each of whose elements are also elements of $\wp(\mathcal{T})$.

Extend maps two times into the set of times that represents the interval between the first time and the second time.

$$\textbf{Extend} : \mathcal{T} \times \mathcal{T} \rightarrow \mathcal{I} \cup \bot$$

$$\textbf{Extend}(t_1,\ t_2) \triangleq \begin{cases} \bot & t_1 > t_2 \\ \{t \mid t_1 \leq t \leq t_2\} & \text{otherwise} \end{cases}$$

Interval maps a set of times into the set of intervals containing the minimum number of non-disjoint intervals represented by the input set. Each time in the input set appears in exactly one interval in the output set and each interval in the output set is itself represented by a set of times.

Interval partitions a set of times into its corresponding set of intervals where each interval is itself represented by a set of times.

$$\textbf{Interval} : \wp(\mathcal{T}) \rightarrow \wp(\mathcal{I}) \cup \emptyset$$

$$\textbf{Interval}(T) \triangleq \begin{cases} \emptyset & T = \emptyset \\ \{I \mid \forall t,\ t \in I, t \in T & \\ \qquad \wedge\ \textbf{Pred}(t) \in T \rightarrow \textbf{Pred}(t) \in I & \text{otherwise} \\ \qquad \wedge\ \textbf{Succ}(t) \in T \rightarrow \textbf{Succ}(t) \in I\} & \end{cases}$$

Note that **Interval** partitions a set of times into the minimum number of non-disjoint intervals represented by the set; each time in T appears in exactly one interval.

B Auxiliary Functions

We used several auxiliary functions in the definition of the historical derivation operator. We present here formal definitions for each of those auxiliary functions.

First takes a set of times from the domain $\wp(\mathcal{T})$ and maps it into the earliest time in the set.

$$\textbf{First} : \wp(\mathcal{T}) \rightarrow \mathcal{T} \cup \bot$$

$$\textbf{First}(T) \triangleq \begin{cases} \bot & T = \emptyset \\ t,\ t \in T \wedge \forall t',\ t' \in T,\ t \leq t' & \text{otherwise} \end{cases}$$

Last takes a set of times from the domain $\wp(\mathcal{T})$ and maps it into the latest time in the set.

$$\textbf{Last} : \wp(\mathcal{T}) \rightarrow \mathcal{T} \cup \bot$$

$$\textbf{Last}(T) \triangleq \begin{cases} \bot & T = \emptyset \\ t,\ t \in T \wedge \forall t',\ t' \in T,\ t \geq t' & \text{otherwise} \end{cases}$$

Pred is the predecessor function on the domain $\mathcal{T}$. It maps a time into its immediate predecessor in the linear ordering of all times.

$$\textbf{Pred} : \mathcal{T} \rightarrow \mathcal{T} \cup \bot$$

$$\textbf{Pred}(t) \triangleq \begin{cases} \bot & t = \textbf{FIRST}(\mathcal{T}) \\ t_P,\ t_P \in \mathcal{T} \wedge t_P < t \wedge \forall t',\ t' \in \mathcal{T} \wedge t' < t,\ t' \leq t_P & \text{otherwise} \end{cases}$$

Succ is the successor function on the domain $\mathcal{T}$. It maps a time into its immediate successor in the linear ordering of all times.

$$\textbf{Succ} : \mathcal{T} \rightarrow \mathcal{T}$$

$$\textbf{Succ}(t) \triangleq t_S,\ t_S \in \mathcal{T} \wedge t_S > t \wedge \forall t',\ t' \in \mathcal{T} \wedge t' > t,\ t' \geq t_S$$

Let the domain $\mathcal{I}$ be the subset of $\wp(\mathcal{T})$ that represents all possible non-disjoint intervals of time.

$$\mathcal{I} \triangleq \{I \mid I \in \wp(\mathcal{T}) \wedge \forall t,\ t \in I \rightarrow \textbf{First}(I) \leq t \leq \textbf{Last}(I)\}$$

t	Element of $\mathcal{T}$
u, v	Temporary variables
V_a	Temporal function in the historical derivation operator
$valid(r(N_a))$	Time-stamp of attribute N_a of tuple r
$valid(r_a)$	Shorthand for $valid(r(N_a))$
$value(r(N_a))$	Value component of attribute N_a of tuple r
$value(r_a)$	Shorthand for $value(r(N_a))$
w	Aggregation window function
$\hat{w}(t)$	window at time t
X	Set of by-list attributes in an aggregate
x, z	Number of tuple variables appearing in an aggregate

A Notational Conventions

This appendix describes the notational conventions used in this paper.

Notation	Usage
$\hat{\cup}$	Historical union operator
$\hat{-}$	Historical difference operator
$\hat{\times}$	Historical cartesian product operator
$\hat{\sigma}$	Historical selection operator
$\hat{\pi}$	Historical projection operator
$\hat{\delta}$	Historical derivation operator
$\widehat{A}$	Historical aggregation function for non-unique aggregates
$\widehat{AU}$	Historical aggregation function for unique aggregates
F	Predicate in the historical selection operator
f	Scalar aggregate
G	Predicate in the historical derivation operator
$\mathcal{I}$	Domain of intervals
I	Interval
I_N	Interval from the time-stamp of attribute N
k	Number of relations
m, m_i	Number of attributes in relation schemes $\mathcal{N}$, $\mathcal{N}_i$
$\mathcal{N}$, $\mathcal{N}_R$	Relation schemes
N, N_a, $N_{i,a}$	Attribute names
n	Length of target list or by-list
$\wp(\mathcal{I})$	Power set of $\mathcal{I}$
$\wp(\mathcal{T})$	Power set of $\mathcal{T}$
p, y	Number of attributes appearing in an aggregate
Q, R, R_i	Historical relations
q, r, r_i	Historical tuple variables
Q', R', R'_i	TQuel relations
q', r', r'_i	TQuel tuple variables
$\mathcal{T}$	Time Domain
T	Subset of $\mathcal{T}$

[Tandem 1983] Tandem Computers, Inc. *ENFORM Reference Manual.* Cupertino, CA, 1983.

[Tansel 1986] Tansel, A.U. *Adding Time Dimension to Relational Model and Extending Relational Algebra. Information Systems*, 11, No. 4 (1986), pp. 343–355.

[Tansel et al. 1989] Tansel, A.U., M.E. Arkun and G. Özsoyoğlu. *Time-By-Example Query Language for Historical Databases. IEEE Transactions on Software Engineering*, 15, No. 4, Apr. 1989, pp. 464–478.

[Ullman 1988] Ullman, J.D. *Principles of Database and Knowledge-Base Systems.* Potomac, Maryland: Computer Science Press, 1988. Vol. 1.

[Wong & Youssefi 1976] Wong, E. and K. Youssefi. *Decomposition - A Strategy for Query Processing. ACM Transactions on Database Systems*, 1, No. 3, Sep. 1976, pp. 223–241.

[Yao 1979] Yao, S.B. *Optimization of Query Evaluation Algorithms. ACM Transactions on Database Systems*, 4, No. 2, June 1979, pp. 133–155.

[Özsoyoğlu et al. 1987] Özsoyoğlu, G., Z. Özsoyoğlu and V. Matos. *Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregate Functions. ACM Transactions on Database Systems*, 12, No. 4, Dec. 1987, pp. 566–592.

[Roth et al. 1988] Roth, Mark A., Henry F. Korth and Abraham Silberschatz. *Extended Algebra and Calculus for Nested Relational Databases. ACM Transactions on Database Systems*, 13, No. 4, Dec. 1988, pp. 389–417.

[Roussopoulos91 1991] Roussopoulos, N. *An Incremental Access Method for ViewCache: Concept, Algorithms, and Cost Analysis. ACM Transactions on Database Systems*, 16, No. 3, september 1991, pp. 535-563.

[Sadeghi 1987] Sadeghi, R. *A Database Query Language for Operations on Historical Data.* PhD. Diss. Dundee College of Technology, Dec. 1987.

[Sarda 1990] Sarda, N. *Algebra and Query Language for a Historical Data Model. The Computer Journal*, 33, No. 1, Feb. 1990, pp. 11–18.

[Schek & Scholl 1986] Schek, H.-J., Scholl, M.H. *The Relational Model with Relation-valued Attributes. Information Systems*, 11, No. 2 (1986), pp. 137–147.

[Selinger et al. 1979] Selinger, P.G., M.M. Astrahan, D.D. Chamberlin, R.A. Lorie and T.G. Price. *Access Path Selection in a Relational Database Management System*, in *Proceedings of ACM SIGMOD International Conference on Management of Data.* Ed. P.A. Bernstein. Association for Computing Machinery. Boston, MA: 1979, pp. 23–34.

[Smith & Chang 1975] Smith, J.M. and P.Y-T. Chang. *Optimizing the Performance of a Relational Algebra Database Interface. Communications of the Association of Computing Machinery*, 18, No. 10, Oct. 1975, pp. 568–579.

[Snodgrass & Ahn 1985] Snodgrass, R. and I. Ahn. *A Taxonomy of Time in Databases*, in *Proceedings of ACM SIGMOD International Conference on Management of Data.* Ed. S. Navathe. Association for Computing Machinery. Austin, TX: May 1985, pp. 236–246.

[Snodgrass & Ahn 1986] Snodgrass, R. and I. Ahn. *Temporal Databases. IEEE Computer*, 19, No. 9, Sep. 1986, pp. 35–42.

[Snodgrass 1987] Snodgrass, R. *The Temporal Query Language TQuel. ACM Transactions on Database Systems*, 12, No. 2, June 1987, pp. 247–298.

[Snodgrass et al. 1989] Snodgrass, R., S. Gomez and E. McKenzie. *Aggregates in the Temporal Query Language TQuel.* Technical Report TR–89–26. Department of Computer Science, University of Arizona. Nov. 1989.

[Stonebraker 1975] Stonebraker, M. *Implementation of Integrity Constraints and Views by Query Modification*, in *Proceedings of ACM SIGMOD International Conference on Management of Data.* Association for Computing Machinery. San Jose, CA: June 1975.

[Stonebraker et al. 1976] Stonebraker, M., E. Wong, P. Kreps and G. Held. *The Design and Implementation of INGRES. ACM Transactions on Database Systems*, 1, No. 3, Sep. 1976, pp. 189–222.

Model for Relational Databases with Transaction Time. IEEE Transactions on Knowledge and Data Engineering (to appear), (1991).

[Jones et al. 1979] Jones, S., P. Mason and R. Stamper. *LEGOL 2.0: A Relational Specification Language for Complex Rules. Information Systems*, 4, No. 4, Nov. 1979, pp. 293–305.

[Klug 1982] Klug, A. *Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions. Journal of the Association of Computing Machinery*, 29, No. 3, July 1982, pp. 699–717.

[Knuth 1973] Knuth, D.E. *Fundamental Algorithms.* Vol. 1, Second Edition of The Art of Computer Programming. Addison-Wesley, 1973.

[Krishnamurthy et al. 1986] Krishnamurthy, R., H. Boral and C. Zaniolo. *Optimization of Nonrecursive Queries*, in *Proceedings of the Conference on Very Large Databases.* Ed. Y. Kambayashi. Kyoto, Japan: Aug. 1986, pp. 128–137.

[Lorentzos & Johnson 1988] Lorentzos, N. and R. Johnson. *Extending Relational Algebra to Manipulate Temporal Data. Information Systems*, 13, No. 3 (1988), pp. 289–296.

[Maier 1983] Maier, D. *The Theory of Relational Databases.* Rockville, MD: Computer Science Press, 1983.

[McKenzie 1986] McKenzie, E. *Bibliography: Temporal Databases. ACM SIGMOD Record*, 15, No. 4, Dec. 1986, pp. 40–52.

[McKenzie & Snodgrass 1987] McKenzie, E. and R. Snodgrass. *Extending the Relational Algebra to Support Transaction Time*, in *Proceedings of ACM SIGMOD International Conference on Management of Data.* Ed. U. Dayal and I. Traiger. Association for Computing Machinery. San Francisco, CA: May 1987, pp. 467–478.

[McKenzie 1988] McKenzie, E. *An Algebraic Language for Query and Update of Temporal Databases.* PhD. Diss. Computer Science Department, University of North Carolina at Chapel Hill, Sep. 1988.

[McKenzie & Snodgrass 1990] McKenzie, E. and R. Snodgrass. *Schema Evolution and the Relational Algebra. Information Systems*, 15, No. 2, June 1990, pp. 207–232.

[McKenzie & Snodgrass 1991B] McKenzie, E. and R. Snodgrass. *An Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. ACM Computing Surveys*, 23, No. 4, Dec. 1991, pp. 501–543.

[Navathe & Ahmed 1989] Navathe, S. B. and R. Ahmed. *A Temporal Relational Model and a Query Language. Information Sciences*, 49 (1989), pp. 147–175.

[Overmyer & Stonebraker 1982] Overmyer, R. and M. Stonebraker. *Implementation of a Time Expert in a Database System. ACM SIGMOD Record*, 12, No. 3, Apr. 1982, pp. 51–59.

[Elmasri & Navathe 1989] Elmasri, R. and S.B. Navathe. *Fundamentals of Database Systems.* Benjamin/Cummings Pub. Co., 1989.

[Enderton 1977] Enderton, H.B. *Elements of Set Theory.* New York, N.Y.: Academic Press, Inc., 1977.

[Gadia 1986] Gadia, S.K. *Toward a Multihomogeneous Model for a Temporal Database*, in *Proceedings of the International Conference on Data Engineering.* IEEE Computer Society. Los Angeles, CA: IEEE Computer Society Press, Feb. 1986, pp. 390–397.

[Gadia 1988] Gadia, S.K. *A Homogeneous Relational Model and Query Languages for Temporal Databases. ACM Transactions on Database Systems*, 13, No. 4, Dec. 1988, pp. 418–448.

[Gadia & Yeung 1988] Gadia, S.K. and C.S. Yeung. *A Generalized Model for a Relational Temporal Database*, in *Proceedings of ACM SIGMOD International Conference on Management of Data.* Association for Computing Machinery. Chicago, IL: June 1988, pp. 251–259.

[Hall 1976] Hall, P.A.V. *Optimization of Single Expressions in a Relational Data Base System. IBM Journal of Research and Development*, 20, No. 3, May 1976, pp. 244–257.

[Hanson 1987A] Hanson, E.N. *A Performance Analysis of View Materialization Strategies*, in *Proceedings of the ACM SIGMOD Annual Conference.* Ed. U. Dayal and I. Traiger. Association for Computing Machinery. San Francisco, CA: ACM Press, May 1987, pp. 440–453.

[Hanson 1987B] Hanson, E.N. *Efficient Support for Rules and Derived Objects in Relational Database Systems.* PhD. Diss. Computer Science Department, University of California at Berkeley, Aug. 1987.

[Held et al. 1975] Held, G.D., M. Stonebraker and E. Wong. *INGRES–A Relational Data Base Management System*, in *Proceedings of the AFIPS National Computer Conference.* Anaheim, CA: AFIPS Press, May 1975, pp. 409–416.

[Horwitz & Teitelbaum 1986] Horwitz, S.B. and T. Teitelbaum. *Generting Editing Environments Based on Relations and Attributes. ACM Transactions on Programming Languages and Systems*, 8, No. 4, Oct. 1986, pp. 577–608.

[IBM 1981] IBM *SQL/Data-System, Concepts and Facilities.* Technical Report GH24-5013-0. IBM. Jan. 1981.

[Jaeschke & Schek 1982] Jaeschke, G. and H.J. Schek. *Remarks on the Algebra of Non First Normal Form Relations*, in *Proceedings of the ACM Symposium on Principles of Database Systems.* 1982.

[Jarke & Koch 1984] Jarke, M. and J. Koch. *Query Optimization in Database Systems. ACM Computing Surveys*, 16, No. 2, June 1984, pp. 111–152.

[Jensen et al. 1991] Jensen, C. S., L. Mark and N. Roussopoulos. *Incremental Implementation*

DC: May 1986, pp. 96–107.

[Ahn & Snodgrass 1989] Ahn, I. and R. Snodgrass. *Performance Analysis of Temporal Queries. Information Sciences*, 49 (1989), pp. 103–146.

[Anderson 1982] Anderson, T.L. *Modeling Time at the Conceptual Level*, in *Proceedings of the International Conference on Databases: Improving Usability and Responsiveness*. Ed. P. Scheuermann. Jerusalem, Israel: Academic Press, June 1982, pp. 273–297.

[Ariav 1986] Ariav, G. *A Temporally Oriented Data Model. ACM Transactions on Database Systems*, 11, No. 4, Dec. 1986, pp. 499–527.

[Ben-Zvi 1982] Ben-Zvi, J. *The Time Relational Model*. PhD. Diss. Computer Science Department, UCLA, 1982.

[Bhargava & Gadia 1989] Bhargava, G. and S.K. Gadia. *A 2-dimensional temporal relational database model for querying errors and updates, and for achieving zero information-loss.* Technical Report TR#89-24. Department of Computer Science, Iowa State University. Dec. 1989.

[Bhargava & Gadia 1991] Bhargava, G. and S.K. Gadia. *Relational database systems with zero information-loss. IEEE Transactions on Knowledge and Data Engineering*, (to appear) (1991).

[Blakeley et al. 1986] Blakeley, J.A., P.-A. Larson and F.W. Tompa. *Efficiently Updating Materialized Views*, in *Proceedings of ACM SIGMOD International Conference on Management of Data*. Ed. C. Zaniolo. Association for Computing Machinery. Washington, DC: May 1986, pp. 61–71.

[Bontempo 1983] Bontempo, C. J. *Feature Analysis of Query-By-Example*, in Relational Database Systems. New York: Springer-Verlag, 1983. pp. 409–433.

[Clifford & Tansel 1985] Clifford, J. and A.U. Tansel. *On an Algebra for Historical Relational Databases: Two Views*, in *Proceedings of ACM SIGMOD International Conference on Management of Data*. Ed. S. Navathe. Association for Computing Machinery. Austin, TX: May 1985, pp. 247–265.

[Clifford & Croker 1987] Clifford, J. and A. Croker. *The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans*, in *Proceedings of the International Conference on Data Engineering*. IEEE Computer Society. Los Angeles, CA: IEEE Computer Society Press, Feb. 1987, pp. 528–537.

[Codd 1970] Codd, E.F. *A Relational Model of Data for Large Shared Data Banks. Communications of the Association of Computing Machinery*, 13, No. 6, June 1970, pp. 377–387.

[Codd 1972] Codd, E.F. *Relational Completeness of Data Base Sublanguages*, in Data Base Systems. Vol. 6 of Courant Computer Symposia Series. Englewood Cliffs, N.J.: Prentice Hall, 1972. pp. 65–98.

the referenced snapshot relations to historical relations by assigning all attributes the same time-stamp. The historical operators $\hat{\cup}$, $\hat{-}$, $\hat{\times}$, $\hat{\sigma}$, and $\hat{\pi}$ reduce to their snapshot counterparts when all attribute time-stamps are identical. The algebra is also consistent with the conceptual view of historical relations as 3-dimensional, space-filling objects and the view of operations on historical relations as "volume" operations. In addition, the algebra supports historical queries, is closed, includes aggregates, does not exhibit temporal data loss as an operator side-effect, and has a unique representation for each historical relation. The algebra satisfies all but one of the commutative, associative, and distributive tautologies involving union, difference, and cartesian product as well as the non-conditional commutative laws involving selection and projection. Additional equivalences involving historical derivation also hold. We discussed representations of historical relations on secondary storage that are straightforward extensions of those of conventional relations. Finally, we have defined an incremental version of the algebra that supports incrementally materialized views of historical relations. This version has been implemented. Hence, we have demonstrated that the algebra may be efficiently implemented.

Secondly, the algebra is shown to have the expressive power of the calculus-based temporal query language TQuel. As such, the algebra provides an executable equivalent of a declarative query language. Because all but one of the equivalences that hold for the snapshot algebra also hold for the historical algebra, most existing optimization algorithms may be naturally extended to optimize historical queries. Conversion between historical relations and the tuple-time-stamping assumed by TQuel is simple and efficient.

The obvious future work is an implementation of the algebra as defined here and development of optimization strategies. At this point, we feel that the formal definition of temporal databases and their query languages has yielded many results (c.f., [McKenzie 1986]), while implementation issues such as access methods, physical storage structures, and novel storage devices remain largely unexplored.

8 Acknowledgements

This research was supported in part by the Office of Naval Research under contract N00014-86-K-0680 and in part by NSF grants DCR-8402339 and IRI-8902707. Research by the first author was sponsored in part by the United States Air Force. Research by the second author was sponsored in part by an IBM Faculty Development Award and contract #1124. We thank Suchen Hsu for her insight into page organization. The suggestions of the referees significantly improved the presentation.

Bibliography

[Ahn & Snodgrass 1986] Ahn, I. and R. Snodgrass. *Performance Evaluation of a Temporal Database Management System*, in *Proceedings of ACM SIGMOD International Conference on Management of Data*. Ed. C. Zaniolo. Association for Computing Machinery. Washington,

relations and operations, and we wished to ensure that there was a unique representation for each relation. Retention of the set-theoretic semantics of the operators would have prevented the algebra from satisfying these criteria. We defined the semantics of the historical version of each snapshot operator to be a consistent extension of the snapshot operator's semantics. Hence, each expression in the snapshot algebra has an equivalent counterpart in the historical algebra and expressions in the historical algebra reduce to their snapshot counterparts when all attribute time-stamps are the same. Also, we defined all operators to prevent loss of temporal information as an operator side-effect.

Jones, Lorentzos, Navathe, and Sarda retain the set-theoretic semantics of the basic relational operators (in Sarda's algebra the selection operator is the one exception). The other algebras extend the semantics of the basic relational operators to handle time.

6.5 New Temporal Operators

We chose to handle temporal selection, projection, and aggregation by introducing new operators ($\hat{\delta}$, $\hat{A}$ and $\widehat{AU}$) to perform these functions. We would have preferred separate operators for temporal selection and projection, but were forced to include both functions in the derivation operator because we chose to allow set-valued attribute time-stamps. We defined the new operators $\widehat{SN}$ and AT to convert between snapshot and historical relations. We also utilize the rollback operators (ρ and $\hat{\rho}$) to accommodate transaction time.

Ben-Zvi includes temporal selection as part of the selection operator. Clifford, Gadia, Navathe, and Sadeghi proved new operators to support temporal selection. Jones, Lorentzos, and Tansel provide temporal selection indirectly through new operators. Ben-Zvi supports a limited version of temporal projection through the projection operator. Tansel provides three new operators to do temporal projection. A limited capability for temporal projection is available in the remaining algebras indirectly through new operators. Ben-Zvi, Jones, Navathe, and Tansel include new operators to do aggregation; the rest do not.

7 Summary

The design of an historical algebra that simultaneously satisfies many desirable properties is a surprisingly difficult task. Since all desirable properties of historical algebras are not compatible [McKenzie & Snodgrass 1991B], the best that can be hoped for is not an algebra with all possible desirable properties but an algebra with a maximal subset of the most desirable properties.

This paper makes two contributions. First, an historical algebra is defined as a straightforward extension of the conventional relational algebra. The historical algebra defined here has what we consider to be the most desirable properties of an historical algebra. Specifically, each relation and algebraic expression in the snapshot algebra has an equivalent counterpart in the historical algebra. Expressions in the snapshot algebra can be converted to their historical equivalent simply by replacing each snapshot operator with its corresponding historical operator and converting

with the semantics of the individual snapshot algebraic operations as operations on 2-dimensional tables, extended to account for the additional dimension represented by valid time. Simultaneously providing this conceptual view, presenting loss of information about temporal relationships as an operator side-effect, and preserving the tautologies listed in Section 5.1 is not possible unless time-stamps are set-valued [McKenzie & Snodgrass 1991B].

The decision to allow set-valued attribute time-stamps unfortunately prevented the algebra from having other less desirable, but nonetheless desirable, properties. If we had not specified set-valued attribute time-stamps, we could have retained the first-normal-form property of the snapshot algebra. Also, we could have replaced the single complex historical derivation operator with two simpler operators, one performing historical selection and the other performing historical projection.

Clifford and Gadia also allow set-valued time-stamps. The other algebras allow only single-valued time-stamps.

6.3 Single-valued Attributes

We decided to restrict attributes to single values to retain in the historical algebra the commutative properties of the selection operator found in the snapshot algebra. If we had allowed set-valued attributes, without imposing intra-tuple restrictions on attribute time-stamps, then we would had to have combined the functions of the selection and historical derivation operators into a single, more complex operator. This consolidation would have been necessary to ensure that the temporal predicate in the current historical derivation operator was considered to be true for an assignment of intervals to attribute names only when the predicate in the current selection operator held for the attribute values associated with those intervals (since multiple input tuples might participate in the computation of a single result tuple). This new operator would have satisfied the commutative properties of the current selection operator only in restricted cases, thereby limiting the usefulness of key optimization strategies.

A second reason for adopting single-valued attributes was to simplify the page layout. In particular, tuples that are fixed-width in a conventional relation continue to have this useful property when time is added, and techniques such as space compression apply without change to historical relations. Also, conversion between this representation and tuple time-stamping is easier than with set-valued attributes.

Ben-Zvi, Jones, Lorentzos, Navathe, and Sadeghi also restrict attributes to single values. The remaining algebras allow set-valued attribute values.

6.4 Extended Operator Semantics

We chose to extend the semantics of the conventional relational operators to handle the temporal dimension directly, rather than define new operators to deal with the temporal dimension. There were two reasons: we wanted to support a three-dimensional conceptual visualization of historical

and evaluation of these historical algebras can be found elsewhere [McKenzie & Snodgrass 1991B].

6.1 Time-stamping Attribute Values

We decided to time-stamp attribute values rather than tuples to support historical queries. We wanted the algebra to allow for the derivation of information valid at a time t from information in underlying relations valid at other times, much as the snapshot algebra allows for the derivation of information about entities or relationships from information in underlying relations about other entities or relationships. This requirement implies that the algebra allow units of related information, possibly valid at disjoint times, to be combined into a single related unit of information possibly valid at some other times. Support for such a capability required that we define a cartesian product operator that concatenates tuples, independent of their valid times, and preserves, in the resulting tuple, the valid-time information for each of the underlying tuples. Only by time-stamping attribute values could we define a cartesian product operator with this property and maintain closure under cartesian product.

Lorentzos, Gadia, and Tansel also time-stamp attribute values. Only Gadia's homogeneous model requires that a tuple's attribute time-stamps be identical; the others allow tuples with disjoint attribute time-stamps. Clifford assigns a time-stamp, termed a *lifespan*, to each tuple in a relation and to each attribute in the relation's scheme. The lifespan of each attribute of a tuple is then computed as the intersection of the tuple's lifespan and the attribute's lifespan, as specified in the relation's scheme. Ben-Zvi, Jones, Navathe, Sadeghi, and Sarda all time-stamp tuples only.

6.2 Set-valued Time-stamps

There are a variety of possible representations for valid time. Time-stamps can correspond to a single chronon, to an interval delimited by two chronons, to sets of non-contiguous intervals, or to sets of chronons. We decided to represent valid time as a set of (not necessarily consecutive) chronons, for two reasons. First, we wanted to ensure a unique representation for each historical relation. If we had decided to disallow set-valued attribute time-stamps, then we would had to have permitted value-equivalent tuples to model accurately real-world temporal relationships. Yet, value-equivalent tuples, because they spread temporal relationships among attributes across tuples, would have caused problems in defining an algebra. If value-equivalent tuples had been allowed (and set-valued attribute time-stamps disallowed), a unique representation for each historical relation could not have been specified without imposing inter-tuple restrictions on the attribute time-stamps of value-equivalent tuples.

Second, we wanted the algebra to support the user-oriented conceptual view of historical relations as 3-dimensional objects [Ariav 1986, Clifford & Tansel 1985], and each historical operator to have an interpretation, consistent with its semantics, in accordance with this conceptual framework, so that historical operators manipulate space-filling objects. For example, the difference operator should take two space-filling objects (i.e., historical relations) and produce a object that represents the mass (i.e., total historical information) present in the first object but not present in the second object. Note that this description of operations on historical relations is also consistent

A different conclusion is reached when considering historical relations. The storage structure may be organized in such a way that updates are more costly than those to a conventional relation by perhaps only a constant factor. However, retrievals are more costly by a factor that is roughly sublinear to linear in the size of the relation [Ahn & Snodgrass 1986, Ahn & Snodgrass 1989]. While update cost remains fairly constant, retrieval costs increases monotonically over time. At some point, probably quite soon, incremental view materialization becomes beneficial for most temporal views.

Hence, it is desirable that the historical algebra be able to support incremental view materialization. We have defined an alternate, incremental semantics for the historical operators. In this semantics, each operator is defined as a mapping from one (or two) relation states and its (their) differential onto a resulting relation state and its corresponding differential [McKenzie 1988]. We have also developed a prototype implementation of this algebra as an existence proof that it could be done. The historical algebra can thus support both unmaterialized views (via query modification [Stonebraker 1975]) and materialized views, and can support various view maintenance strategies, such as in-line view evaluation, immediate-recomputed materialization, and immediate-incremental materialization. In concert with techniques developed for rollback relations [Jensen et al. 1991], it can also support these maintenance strategies for views defined on temporal relations that incorporate both valid and transaction time.

In summary, we have shown that it is possible to implement the algebra efficiently.

6 Review of Design Decisions

In defining the objects in the historical algebra, we were faced with three major design decisions: (i) whether to time-stamp tuples or attribute values, (ii) whether to allow single-valued or set-valued time-stamps, and (iii) whether to allow single-valued or set-valued attributes. To extend operations in the snapshot algebra to handle valid time, we had to make two subsequent design decisions. (iv) Is the set-theoretic semantics of the basic relational operators retained and new operators introduced to deal with the temporal dimension of the real-world phenomena being modeled, or is the semantics of the existing relational operators extended to account for the temporal dimension directly? If the latter, then (v) how do these operators compute the valid time of attributes in resulting tuples? In particular, how does the algebra handle *temporal selection* (i.e., tuple selection based on valid times), *temporal projection* (i.e., computation of new valid times for a tuple's attributes from their current valid times), and *temporal aggregation* (i.e., computation of a distribution of aggregate values over time); operations that are unique to a historical algebra? We discuss here our choices and the importance of those choices in determining the properties of the algebra. We also mention the choices to these design decisions made by the developers of ten other historical algebras: Ben-Zvi's Time Relational Model [Ben-Zvi 1982], Clifford's proposed extension to the snapshot algebra [Clifford & Croker 1987], Gadia's historical algebras [Bhargava & Gadia 1989, Bhargava & Gadia 1991, Gadia 1986, Gadia 1988], Jones' extension to the snapshot algebra to support time-oriented operations for LEGOL [Jones et al. 1979], Lorentzos' Temporal Relational Algebra [Lorentzos & Johnson 1988], Navathe's historical algebra [Navathe & Ahmed 1989], Sadeghi's algebra [Sadeghi 1987], Sarda's algebra [Sarda 1990], and Tansel's historical algebra [Tansel 1986]. A detailed review

There are a variety of ways to effect this transformation. The brute-force method is to first cluster the relation on a key, perhaps by sorting the relation, so that all of the versions are collected on the same page, then link up the intervals, distributing them to the attributes. Since redundant attribute values occur in a tuple time-stamped representation, the space requirements will decrease during this conversion, guaranteeing that no new overflow pages will result. If we record in the schema that all attributes contain the same time-stamp, then we need not duplicate interval lists for each attribute. The conversion can even be done in parallel with any of the historical operators. When the operator fetches another tuple, the interval list can be constructed and passed to the operator, assuming that the underlying relation was clustered on the key.

Once an algebraic expression has computed a result relation, it must be converted back into a tuple time-stamped representation. This step is even easier than the other direction. The TQuel semantics presented in Section 4.2 ensures that the time-stamps of all of the attributes are identical within a tuple. So all that is necessary is to make a duplicate each tuple for each interval in the interval list. This expansion also can be done within any of the historical operators. The conversion is similar to the *Unpack* operation (also termed *unnest*) in non-1NF relations. It has been shown that applying the Pack operation followed by Unpack operation, i.e., performing the empty algebraic expression on a tuple-time-stamped relation, produces the original relation [Jaeschke & Schek 1982].

Finally, there is no reason why a relation *logically* time-stamped on a tuple basis with single intervals can't be stored *physically* as time-stamped with a set (linked list) of intervals, in concert with the space optimization of utilizing only one interval pointer for the entire tuple. This storage structure requires conversion only on display, which is much less time-critical than conversion on access and on storage.

5.3 Incremental Execution

A promising approach to achieve greater efficiency in temporal DBMS's is that of incremental view materialization [Blakeley et al. 1986, Hanson 1987A, Hanson 1987B, Horwitz & Teitelbaum 1986, Roussopoulos91 1991]. This process brings the view up-to-date following the update of one of its underlying relations by identifying the tuples that must be inserted into, and the tuples that must be deleted from, the view's old state for the view's new state to be consistent with the new states of its underlying relations, without having to recompute the view itself. The net changes that an update operation makes to a stored relation, either a base relation or a materialized view, is termed the relation's *differential*.

Incremental view materialization is more efficient than processing without views if four conditions are satisfied simultaneously: (1) the number of queries against a view is sufficiently higher than the number of updates to its underlying relations, (2) the sizes of the underlying relations are sufficiently large, (3) the selectivity factor of the view predicate is sufficiently low, and (4) the percentage of the view retrieved by queries is sufficiently high [Roussopoulos91 1991]. Since these conditions are rather restrictive in practice, commercial DBMS's do not support incremental view materialization.

snapshot algebra also hold for the historical algebra, the search space of equivalent query plans for a historical query should be comparable in size to that for an analogous snapshot query. Hence, the historical algebra does not limit the practical use of query plan transformation as an optimization technique for historical queries. Also, most algorithms for optimization of snapshot queries may be extended to optimize historical queries by taking into account the possible presence of historical derivation operators in query plans.

5.2 Page Structure

An historical tuple is more complex than a conventional tuple, because time-stamps are sets. As first normal form (1NF) dictates that each value of a tuple be atomic [Elmasri & Navathe 1989], historical relations cannot be considered to be in 1NF. However, they are close, in that the value component of an attribute *is* atomic. One simple means of retaining much of the simplicity of conventional relations is to implement the set of chronons forming the time-stamp of an attribute as a linked list of intervals, each represented with an *interval cell* containing a starting time-stamp, an ending time-stamp, and a pointer to the next interval. An attribute's time-stamp then becomes a fixed-length pointer field. For page sizes under 4K bytes, a single byte sufficies for a pointer; if overflow pages are permitted then two bytes are required for the pointer. Using interval lists, fixed-length tuples remain of fixed length even when time-stamps are added, and conventional techniques, e.g., of attribute-value space compression and null value representation, still apply.

Various space management approaches are available to contend with the interval lists now present. If tuples are fixed-length, then the page may be partitioned into fixed-length slots, each to be occupied either by a tuple or by several interval cells. Variable-length tuples are often handled by placing the tuples at the top of the page growing down and tuple headers at the bottom of the page growing up, with free space in the middle [Stonebraker et al. 1976]. The interval lists also vary in size. They can either be allocated in the same space as the tuples, or the tuple headers can be pre-allocated (since they are short, 1–2 bytes, preallocation will not waste much space), and the intervals can start at the bottom of the page and grow up. In all cases, compaction will be necessary upon deallocation of an interval [Knuth 1973].

The time-stamps for time-invariant attributes may be either stored as a special value, distinguishable from an interval pointer, that represents the set containing all chronons, or not stored at all, but instead indicated as time-invariant in the schema. Several attributes often share the same time-stamp; again, this can be indicated in the schema, with only one interval pointer allocated for the group (this implementation shares some aspects with Gadia's multi-homogeneous data model [Gadia 1986]), or can be represented at the extension level by having multiple interval pointers pointing to the same interval list head cell (though care must be taken when modifying such shared interval lists).

If the algebra is used to implement TQuel, then a conversion will be necessary between tuple time-stamping, where each tuple is associated with a single interval, and attribute-value time-stamping, in which each attribute is associated with potentially multiple intervals. This conversion is formalized in the transformation function **T** discussed in Section 4.2; it is similar to the *Pack* operation (also termed *nest*) proposed for non-1NF relations [Tansel 1986, Özsoyoğlu et al. 1987].

$$\hat{\delta}_{G,\,V_1,\,...,\,V_{m_q}}(\hat{\delta}_{G,\,I_1,\,...,\,I_{m_q}}(Q)) \equiv \hat{\delta}_{G,\,V_1,\,...,\,V_{m_q}}(Q)$$

$$\hat{\delta}_{G_1,\,I_1,\,...,\,I_{m_q}}(\hat{\delta}_{G_2,\,I_1,\,...,\,I_{m_q}}(Q)) \equiv \hat{\delta}_{G_2,\,I_1,\,...,\,I_{m_q}}(\hat{\delta}_{G_1,\,V_1,\,...,\,V_{m_q}}(Q))$$

$\hat{\delta}$ is also commutative with selection.

$$\hat{\delta}_{G,\,V_1,\,...,\,V_{m_q}}(\hat{\sigma}_F(Q)) \equiv \hat{\sigma}_F(\hat{\delta}_{G,\,V_1,\,...,\,V_{m_q}}(Q))$$

If the temporal predicate G references only attributes of Q, then

$$\hat{\delta}_{G,\,q_1,\,...,\,r_{m_r}}(Q\,\hat{\times}R) \equiv \hat{\delta}_{G,\,q_1,\,...,\,q_{m_q}}(Q)\,\hat{\times}\,R.$$

If G can be expressed as $G_1 \wedge G_2$, where G_1 references only attributes of Q and G_2 references only attributes of R, then

$$\hat{\delta}_{G,\,q_1,\,...,\,r_{m_r}}(Q\,\hat{\times}R) \equiv \hat{\delta}_{G_1,\,q_1,\,...,\,q_{m_q}}(Q)\,\hat{\times}\,\hat{\delta}_{G_2,\,r_1,\,...,\,r_{m_r}}(R).$$

If G_1 references only attributes of Q but G_2 references attributes of Q and R, then

$$\hat{\delta}_{G,\,q_1,\,...,\,r_{m_r}}(Q\,\hat{\times}R) \equiv \hat{\delta}_{G_2,\,q_1,\,...,\,r_{m_r}}(\hat{\delta}_{G_1,\,q_1,\,...,\,q_{m_q}}(Q)\,\hat{\times}\,R).$$

These conditional equivalences involving historical derivation are important because they can be used to move temporal selection before cartesian product in a query plan transformation. The above equivalences imply that if G can be expressed as $G_1 \wedge G_2$, where G_1 references only attributes of Q and G_2 references only attributes of R, then

$$\hat{\delta}_{G,\,V_{q,1},\,...,\,V_{r,m_r}}(Q\,\hat{\times}R) \equiv \hat{\delta}_{G,\,V_{q,1},\,...,\,V_{r,m_r}}(\hat{\delta}_{G_1,\,q_1,\,...,\,q_{m_q}}(Q)\,\hat{\times}\,\hat{\delta}_{G_2,\,r_1,\,...,\,r_{m_r}}(R)).$$

Performing the temporal selection function *twice* may be cost effective, depending on the size of Q and R and the selectivity of the predicates G_1 and G_2. Note that no equivalences are presented that involve historical derivation and union, difference, or projection: historical derivation doesn't commute with projection or distribute over union or difference, even conditionally, as these operators may change attribute time-stamps.

In summary, all the above non-conditional and conditional equivalences may be used, along with statistical descriptions of historical databases and cost models for query plan execution, to optimize individual historical queries. Because all but one of the equivalences that hold for the

$$Q = \{\ \langle(\text{Norman},\ \{1,2,3\}),\ (\text{Texas},\ \{1,2,3\})\rangle\ \}$$
$$R = \{\ \langle(\text{Norman},\ \{1,2\}),\ (\text{English},\ \{1,2\})\rangle\ \}$$
$$S = \{\ \langle(\text{Norman},\ \{2\}),\ (\text{English},\ \{2\})\rangle\ \}$$

Then,

$$Q \hat{\times} (R \hat{-} S) = \{\ \langle(\text{Norman},\ \{1,2,3\}),\ (\text{Texas},\ \{1,2,3\}),\ (\text{Norman},\ \{1\}),\ (\text{English},\ \{1\})\rangle\ \}$$
$$(Q \hat{\times} R) \hat{-} (Q \hat{\times} S) = \{\ \langle(\text{Norman},\ \emptyset),\ (\text{Texas},\ \emptyset),\ (\text{Norman},\ \{1\}),\ (\text{English},\ \{1\})\rangle\ \}$$ ∎

Ullman identifies several conditional equivalences involving selection and projection that can be used in optimizing snapshot queries [Ullman 1988]. These conditional equivalences also hold in the historical algebra (again, the proofs are cumbersome and unenlightening). We list these equivalences here, along with their accompanying conditions.

- If the non-temporal predicate F references only attributes of Q, then $\hat{\sigma}_F(Q \hat{\times} R) \equiv \hat{\sigma}_F(Q) \hat{\times} R$.
- If F can be expressed as $F_1 \wedge F_2$, where F_1 references only attributes of Q and F_2 references only attributes of R, then $\hat{\sigma}_F(Q \hat{\times} R) \equiv \hat{\sigma}_{F_1}(Q) \hat{\times} \hat{\sigma}_{F_2}(R)$.
- If F_1 references only attributes of Q but F_2 references attributes of Q and R, then $\hat{\sigma}_F(Q \hat{\times} R) \equiv \hat{\sigma}_{F_2}(\hat{\sigma}_{F_1}(Q) \hat{\times} R)$.
- If F references only attributes in the set X of projection attributes, then $\hat{\pi}_X(\hat{\sigma}_F(Q)) \equiv \hat{\sigma}_F(\hat{\pi}_X(Q))$.
- If F also references attributes X' that are not in the set X of projection attributes, then $\hat{\pi}_X(\hat{\sigma}_F(Q)) \equiv \hat{\pi}_X(\hat{\sigma}_F(\hat{\pi}_{X \cup X'}(Q)))$.
- If X_1 and X_2 are sets of projection attributes where $X_1 \subseteq X_2$, then $\hat{\pi}_{X_1}(\hat{\pi}_{X_2}(Q)) \equiv \hat{\pi}_{X_1}(Q)$.
- If X is a set of projection attributes where X_q are attributes of Q, X_r are attributes of R, and $X_q \cup X_r = X$, then $\hat{\pi}_X(Q \hat{\times} R) \equiv \hat{\pi}_{X_q}(Q) \hat{\times} \hat{\pi}_{X_r}(R)$.

In addition to the conditional equivalences involving selection and projection, several conditional equivalences involving historical derivation, which have no snapshot counterparts, hold for the historical algebra. For these equivalences, recall from the definition of historical derivation on page 10 that

$$\hat{\delta}_{G,\, I_1,\, \ldots,\, I_{m_q}}(Q)$$

is a special form of the derivation operator that performs only the temporal selection function. Because this special form of historical derivation has properties analogous to those of non-temporal selection, the following equivalences involving historical derivation hold.

argue elsewhere that this equivalence is not a desirable property of historical algebras [McKenzie & Snodgrass 1991B]). The algebra also supports all the non-conditional commutative and distributive laws involving selection and projection presented by Ullman [Ullman 1988]. Finally, the algebra supports the commutative law of historical selection and historical derivation.

For the theorems that follow assume that Q, R, and S are historical relations.

Theorem 9 *The following equivalences hold for the historical algebra.*

$$Q \hat{\cup} R \equiv R \hat{\cup} Q \quad (1)$$

$$Q \hat{\times} R \equiv R \hat{\times} Q \quad (2)$$

$$\hat{\sigma}_{F_1}(\hat{\sigma}_{F_2}(Q)) \equiv \hat{\sigma}_{F_2}(\hat{\sigma}_{F_1}(Q)) \quad (3)$$

$$Q \hat{\cup} (R \hat{\cup} S) \equiv (Q \hat{\cup} R) \hat{\cup} S \quad (4)$$

$$Q \hat{\times} (R \hat{\times} S) \equiv (Q \hat{\times} R) \hat{\times} S \quad (5)$$

$$Q \hat{\times} (R \hat{\cup} S) \equiv (Q \hat{\times} R) \hat{\cup} (Q \hat{\times} S) \quad (6)$$

$$\hat{\sigma}_F(Q \hat{\cup} R) \equiv \hat{\sigma}_F(Q) \hat{\cup} \hat{\sigma}_F(R) \quad (7)$$

$$\hat{\sigma}_F(Q \hat{-} R) \equiv \hat{\sigma}_F(Q) \hat{-} \hat{\sigma}_F(R) \quad (8)$$

$$\hat{\pi}_X(Q \hat{\cup} R) \equiv \hat{\pi}_X(Q) \hat{\cup} \hat{\pi}_X(R) \quad (9)$$

PROOF. The proofs of the first two equivalences follow directly from the definitions of historical union and historical cartesian product. For the third equivalence, consider the left-hand side of the equivalence. From the definition of historical selection on page 6, we have that a tuple q is in $\hat{\sigma}_{F_1}(\hat{\sigma}_{F_2}(Q))$ if, and only if, $F_1(q) \wedge q \in \hat{\sigma}_{F_2}(Q)$, which implies that q is in $\hat{\sigma}_{F_1}(\hat{\sigma}_{F_2}(Q))$ if, and only if, $F_1(q) \wedge F_2(q) \wedge q \in Q$. Now consider the right-hand side of the equivalence. Again from the definition of historical selection, we have that a tuple q is in $\hat{\sigma}_{F_2}(\hat{\sigma}_{F_1}(Q))$ if, and only if, $F_2(q) \wedge q \in \hat{\sigma}_{F_1}(Q)$, which implies that q is in $\hat{\sigma}_{F_2}(\hat{\sigma}_{F_1}(Q))$ if, and only if, $F_2(q) \wedge F_1(q) \wedge q \in Q$. Hence, the two expressions are shown to denote the same relation. Proofs for the other equivalences, although more notationally cumbersome, can be constructed in a similar fashion. ∎

Theorem 10 *The distributive property of cartesian product over difference does not hold for the historical algebra.*

$$Q \hat{\times} (R \hat{-} S) \not\equiv (Q \hat{\times} R) \hat{-} (Q \hat{\times} S)$$

PROOF. We give an example when the equality does not hold.

used in Theorem 4. ∎

Theorem 7 *The language formed by embedding the historical algebra in the commands used to support transaction time has the expressive power of TQuel.*

This follows directly from the correspondence theorems presented above.

Theorem 8 *The historical algebra defined here is strictly more powerful than TQuel.*

PROOF. For two historical relations R_1' and R_2' with at least two tuples that differ in their time-stamps, consider the algebraic expression $\mathbf{T}(R_1') \hat{\times} \mathbf{T}(R_2')$. Because the semantics of TQuel requires that all attributes within a tuple be associated with identical valid times, this algebraic expression has no counterpart in TQuel. ∎

5 Implementing the Algebra

An historical algebra is a critical part of a DBMS that supports time-varying information. Such an algebra can serve as (1) an appropriate target for a temporal query language processor, (2) an appropriate structure on which to perform optimization, and (3) an appropriate executable formalism for the DBMS to interpret to execute queries. The previous section showed that the historical algebra has the expressive power of TQuel, thus satisfying the first objective just listed. In this section we discuss the other two objectives, focusing in turn on query optimization, page structure, and incremental update of materialized views.

5.1 Query Optimization

Query optimization concerns the problem of selecting the most efficient query plan for a query from the set of all its possible query plans. This problem for snapshot queries has been studied extensively and heuristic algorithms have been proposed for selection of a near optimal query plan based on a statistical description of the database and a cost model for query plan execution [Hall 1976, Jarke & Koch 1984, Krishnamurthy et al. 1986, Selinger et al. 1979, Smith & Chang 1975, Stonebraker et al. 1976, Wong & Youssefi 1976, Yao 1979].

One important aspect of local query optimization is the transformation of one query plan into an equivalent, but more efficient, query plan. The size of the search space of equivalent query plans for a snapshot query is determined in part by the algebraic equivalences available in the snapshot algebra. Both Ullman and Maier identify equivalences that are available in the snapshot algebra for query plan transformation and describe their usefulness to query optimization [Maier 1983, Ullman 1988]. The historical algebra supports all but one of the commutative, associative, and distributive equivalences involving only union, difference, and cartesian product in set theory [Enderton 1977]. The algebra does not support the distributive property of cartesian product over difference. (We

$$R = \hat{\pi}_{N_{i_1,a_1},\ldots,N_{i_n,a_n}}(\hat{\delta}_{\Gamma_\tau,\,\mathbf{Extend}(\Phi_\upsilon,\mathbf{Pred}(\Phi_\chi))} \cap N_{\bar{\jmath}_1,1} \cap \cdots \cap N_{\bar{\jmath}_x,1} \cap N_{agg_1,p}($$
$$\hat{\sigma}_{\Psi_\psi \wedge N_{j_2,c_2}=N_{agg_1,1} \wedge \cdots \wedge N_{j_p,c_p}=N_{agg_1,p-1}}(\mathbf{T}(R'_1)\hat{\times}\cdots\hat{\times}\mathbf{T}(R'_k)\hat{\times}R_{agg_1})))$$

where the reference to f'_1 in τ is replaced by a reference to $N_{agg_1,p+1}$. If the aggregate f'_1 is in υ or χ rather than τ, analogous changes would be required.

4.3.6 Nested Aggregation

The approach described above for handling aggregates in the inner where and when clauses can be used to handle aggregates in a qualifying where or when clause of an aggregate in the outer where, when, or valid clauses. This method of converting TQuel aggregates to their algebraic equivalents, when there is an aggregate in a qualifying clause, can also handle an arbitrary level of nesting of aggregates.

4.4 Correspondence Theorems

Now that all possible locations for aggregates in a TQuel retrieve statement have been dealt with, we have provided algebraic equivalents for all possible TQuel retrieve statements.

Theorem 5 *Every TQuel retrieve statement has an equivalent expression in the historical historical algebra.*

PROOF OUTLINE. Induct on the number of aggregates appearing in the statement to arrive at an equivalent algebraic expression, applying the replacements discussed above in Sections 4.2 and 4.3, as appropriate. Incorporate the handling of transaction time via the rollback operator ($\hat{\rho}$) as discussed elsewhere [McKenzie & Snodgrass 1990]. Construct a tuple calculus expression for the retrieve statement and the algebraic expression, then demonstrate equivalence using the technique used in the proof of Theorem 4. While the proof is aided by the presence of auxiliary relations in the tuple calculus semantics for aggregates [Snodgrass et al. 1989], it is still cumbersome and offers little additional insight. ∎

In a similar fashion, by also using the `modify_state` and `modify_scheme` commands described elsewhere [McKenzie & Snodgrass 1990], one can construct equivalent algebraic statements for the TQuel create, delete, append, replace, and destroy statements, as are given elsewhere [McKenzie 1988].

Theorem 6 *Every TQuel modification statement has an equivalent transaction in the augmented algebra.*

PROOF OUTLINE. Construct a tuple calculus expression for each modification statement [McKenzie 1988] and for its corresponding algebraic expression. Then prove equivalence using the technique

$$
\begin{aligned}
R_{agg_2} = \hat{\sigma}_{N_{agg_2,\,y}=N_{agg_2,\,y+1}}(\widehat{A}_{\text{SMALLEST}\,N_{g_1,\,d_1},\,\Omega_{\omega_2},\,N_{position},\,\{N_{g_2,\,d_2},\,...,\,N_{g_y,\,d_y}\}}(\\
\hat{\delta}_{\Gamma_{\tau_2}\,\wedge\,N_{g_1,\,d_1}=N_{position},\,N_{position},\,N_{\bar{g}_1,\,1},\,...,\,N_{\bar{g}_z,\,m_{\bar{g}_z}}}(\\
\hat{\sigma}_{\Psi_{\psi_2}}(R_{position}\hat{\times}\mathbf{T}(R'_{\bar{g}_1})\hat{\times}\cdots\hat{\times}\mathbf{T}(R'_{\bar{g}_z})))) \\
\hat{\times}\,(R_{position}\,\hat{\cup}\,\{\,\langle\,(0,\,\mathcal{T})\,\rangle\,\})\,, \\
\hat{\pi}_{N_{position},\,N_{g_2,\,d_2},\,...,\,N_{g_y,\,d_y}}(R_{position}\hat{\times}\mathbf{T}(R'_{\bar{g}_1})\hat{\times}\cdots\hat{\times}\mathbf{T}(R'_{\bar{g}_z})))
\end{aligned}
\tag{14}
$$

over the scheme $\mathcal{N}_{agg_2} = \{N_{agg_2,1},\ \ldots,\ N_{agg_2,\,y+1}\}$ where

$$
R_{position} = \hat{\sigma}_{N_{position}\neq 0}(\widehat{A}_{\text{POSITION},\,\infty,\,N_{g_1,\,d_1},\,\emptyset}(\mathbf{T}(R'_{\bar{g}_1}),\ \mathbf{T}(R'_{\bar{g}_1}))) \tag{15}
$$

Expression (14), while structurally equivalent to expression (8) on page 36, is considerably more complex because of the presence of by, when, and where clauses in the nested aggregate. The attributes of $\widehat{A}$'s first argument now include the attributes appearing in the by clause and the attributes of $\widehat{A}$'s second argument include the attributes of relations associated with tuple variables appearing in the aggregate. Also, tuples in the second argument are now required to satisfy the where predicate and, for some interval in the time-stamp of attribute N_{g_1,d_1}, the when predicate. Finally, because TQuel assumes `earliest` and `latest` return $\mathcal{T}$ for an empty partition of R', the tuple $\langle\,(0,\,\mathcal{T})\,\rangle$ is added to $R_{position}$ so that $\mathcal{T}$ will be considered the earliest interval at those times when the partition of $\widehat{A}$'s second argument is empty. Recall that SMALLEST, defined on page 36, returns zero when passed an empty relation.

4.3.4 The Outer Where Clause

Assume that the TQuel aggregate f'_1 appears in ψ in (10) rather than in the target list. Then, the algebraic equivalent of the TQuel retrieve statement is

$$
\begin{aligned}
R = \hat{\pi}_{N_{i_1,\,a_1},\,...,\,N_{i_n,\,a_n}}(\hat{\delta}_{\Gamma_\tau,\,\mathbf{Extend}(\Phi_\upsilon,\,\mathbf{Pred}(\Phi_\chi))\,\cap\,N_{\bar{\jmath}_1,\,1}\,\cap\,\cdots\,\cap\,N_{\bar{\jmath}_x,\,1}\,\cap\,N_{agg_{1,\,p}}}(\\
\hat{\sigma}_{\Psi_\psi\,\wedge\,N_{j_2,\,c_2}=N_{agg_{1,\,1}}\,\wedge\,\cdots\,\wedge\,N_{j_p,\,c_p}=N_{agg_{1,\,p-1}}}(\mathbf{T}(R'_1)\hat{\times}\cdots\hat{\times}\mathbf{T}(R'_k)\hat{\times}R_{agg_1})))
\end{aligned}
$$

where the reference to f'_1 in ψ is replaced by a reference to $N_{agg_1,p}$. Note that the only other change from expression (11) is the elimination of attribute $N_{agg_1,p}$ from the projection, since the aggregate does not appear in the target list.

4.3.5 The Outer When Clause

Assume now that the aggregate f'_1 appears in τ in (10). Then, the algebraic equivalent of the TQuel retrieve statement is

$$
\begin{aligned}
R_{agg_1} = \hat{\pi}_{N_{j_2,\,c_2},\,...,\,N_{j_p,\,c_p},\,N_{agg_1}}(\widehat{A}_{f_1,\,\Omega_{\omega_1},\,N_{j_1,\,c_1},\,\{N_{j_1,\,m_{j_1+1}},\,N_{j_2,\,c_2},\,...,\,N_{j_p,\,c_p}\}}(\\
\hat{\pi}_{N_{\bar{j}_1,\,1},\,...,\,N_{\bar{j}_1,\,m_{\bar{j}+1}},\,N_{\bar{j}_2,\,1},\,...,\,N_{\bar{j}_x,\,m_{\bar{j}_x}}}(\\
\hat{\delta}_{\Gamma_{\tau_1},\,N_{\bar{j}_1,\,1},\,...,\,N_{\bar{j}_1,\,m_{\bar{j}_1}},\,N_{\bar{j}_1,\,m_{\bar{j}_1+1}}\,\cap\,N_{agg_2,\,y},\,N_{\bar{j}_2,\,1},\,...,\,N_{\bar{j}_x,\,m_{\bar{j}_x}},\,N_{agg_2,\,1},\,...,\,N_{agg_2,\,y}}(\\
\hat{\sigma}_{\Psi_{\psi_1}\,\wedge\,N_{g_2,\,d_2}=N_{agg_2,\,1}\,\wedge\,\cdots\,\wedge\,N_{g_y,\,d_y}=N_{agg_2,\,y-1}}(\\
\mathbf{T}(R'_{\bar{j}_1})\hat{\times}\{\,\langle\,(1,\mathcal{T})\,\rangle\,\}\hat{\times}\cdots\hat{\times}\mathbf{T}(R'_{\bar{j}_x})\hat{\times}R_{agg_2})))), \\
\hat{\pi}_{N_{j_1,\,c_1},\,N_{j_1,\,m_{j_1+1}},\,N_{j_2,\,c_2},\,...,\,N_{j_p,\,c_p}}(\mathbf{T}(R'_{\bar{j}_1})\hat{\times}\{\,\langle\,(1,\mathcal{T})\,\rangle\,\}\hat{\times}\cdots\hat{\times}\mathbf{T}(R'_{\bar{j}_x})))
\end{aligned}
\tag{13}
$$

where the attribute name N_{agg_1} here refers to the aggregate produced in $\widehat{A}$ by f_1, the reference to the aggregate f'_2 in ψ_1 is replaced by a reference to $N_{agg_2,\,y}$, and

$$
\begin{aligned}
R_{agg_2} = \widehat{A}_{f_2,\,\Omega_{\omega_2},\,N_{g_1,\,d_1},\,\{N_{g_2,\,d_2},\,...,\,N_{g_y,\,d_y}\}}(\hat{\delta}_{\Gamma_{\tau_2},\,N_{\bar{g}_1,\,1},\,...,\,N_{\bar{g}_z,\,m_{\bar{g}_z}}}(\hat{\sigma}_{\Psi_{\psi_2}}(\mathbf{T}(R'_{\bar{g}_1})\hat{\times}\cdots\hat{\times}\mathbf{T}(R'_{\bar{g}_z}))) \\
\hat{\pi}_{N_{g_1,\,d_1},\,...,\,N_{g_y,\,d_y}}(\mathbf{T}(R'_{\bar{g}_1})\hat{\times}\cdots\hat{\times}\mathbf{T}(R'_{\bar{g}_z})))
\end{aligned}
$$

over the scheme $\mathcal{N}_{agg_2} = \{N_{agg_2,1},\ \ldots,\ N_{agg_2,y}\}$, and f_2 is the family of scalar aggregates corresponding to the family of TQuel aggregates f'_2.

$\{\,\langle\,(1,\mathcal{T})\,\rangle\,\}$ is a constant relation containing a single tuple whose value component may be an arbitrary value from an arbitrary domain. Here, we effectively add an additional attribute to $R_{\bar{j}_1}$ and then use the attribute as an implicit by-list attribute to restrict tuples in the partition of $\mathbf{T}(R'_{\bar{j}_1})\hat{\times}\cdots\hat{\times}\mathbf{T}(R'_{\bar{j}_x})$ at time t to only those tuples that satisfy the predicate in ψ_1 involving the aggregate f'_2 at time t.

4.3.3 The Inner When Clause

Assume now that the aggregate f'_2 appears in τ_1 in (11) rather than in ψ_1. The only aggregates that can appear in τ_1 are `earliest` and `latest`. Therefore, if we let R_{agg_2} be the two-attribute algebraic equivalent of f'_2, then the algebraic equivalent of f'_1 would be the same as that given in (13) for an aggregate in the inner where clause, with one exception. The reference to f'_2 in τ_1 is replaced by a reference to $N_{agg_2,\,y+1}$, not $N_{agg_2,\,y}$. The valid component of $N_{agg_2,\,y}$ is the time when the valid component of $N_{agg_2,\,y+1}$ was the oldest interval, hence $N_{agg_2,\,y+1}$ is used in evaluating τ_1.

If we assume that f'_2 is `earliest`, then R_{agg_2} is

of the $\hat{\delta}$ operator. Also, if $\bar{j}_1$ appears neither outside the aggregate f_1' in (10) nor in its by clause, then R_{agg_1} is replaced by

$$R_{agg_1} \hat{\cup} \{ \langle (\textbf{NULLVALUE}(N_{\bar{j}_1,1}), \{t \mid \forall r \in R_{agg_1} \ (r \notin valid(r(N_{agg_1,p})))\}) \rangle \}.$$

The first change removes the restriction that the valid time of a tuple in the derived relation must intersect the valid time of at least one tuple in the base relation associated with tuple variable $\bar{j}_u$. The second change, ensures that a value (possibly a distinguished null value) for the aggregate is specified at each time $t \in \mathcal{T}$.

4.3.2 The Inner Where Clause

Aggregates may also appear in the where, when, and valid clauses of a TQuel retrieve statement. We now show the algebraic equivalents of TQuel retrieve statements with aggregates in these clauses, first presenting the algebraic equivalent of a TQuel retrieve statement with an aggregate in an inner where clause. Assume that a TQuel aggregate f_2' appears in ψ_1 in (10) and let

$g_1, g_2, \ldots, g_y$ be integers, not necessarily distinct, in the range 1 to k, indicating the (possibly repeated) tuple variables appearing in the nested aggregate where $\forall g_u, \ 1 \leq u \leq y, \ \exists j_v, \ 1 \leq v \leq p, \ g_u = j_v$;

$d_l, \ 1 \leq l \leq y$, be an integer in the range 1 to m_{g_l}, indicating the attribute names appearing in the nested aggregate where $(\forall u)(\forall v), \ (1 \leq u \leq y \wedge 1 \leq v \leq y \wedge u \neq v \wedge g_u = g_v), d_u \neq d_v$; and

$\bar{g}_1, \bar{g}_2, \ldots, \bar{g}_z$ be the distinct integers in $g_1, g_2, \ldots, g_y$ where $\bar{g}_1 = g_1$, indicating the z (non-repeated) tuple variables in the aggregate.

Then, f_2' in ψ_1 has the following syntax

$f_2'($`(`$r'_{g_1}.N_{g_1,d_1}$ `by` $r'_{g_2}.N_{g_2,d_2}$`, ...,` $r'_{g_y}.N_{g_y,d_y}$
`for` ω_2
`where` ψ_2
`when` τ_2`)`

As this TQuel retrieve statement is complicated, containing a nested aggregate with a full complement of by, for, where, and when clauses, we should expect a somewhat complicated algebraic equivalent.

When modified to account for f_2' in ψ_1, the algebraic equivalent of f_1', given in (12), becomes,

The for clause specifies an aggregation window function for the aggregate f_1'. ω_1 contains one or more keywords that determine, along with the time granularity of R_1', ..., R_k', the length of the aggregation window at each time t. The keywords `each instant` represent the aggregation window function $w(t) = 0$ (i.e., an instantaneous aggregate) and the keyword `ever` represents the aggregation window function $w(t) = \infty$ (i.e., a cumulative aggregate). The length of the aggregation window specified by other keywords (e.g., `each day`, `each week`, `each year`) is a function of the underlying time granularity of the database. For example, if the time granularity is a day, then $\omega =$ `each week` translates to the aggregation window function $w(t) = 6$. Also, the aggregation window function need not be a constant function. For example, if the time granularity is a day, then $\omega =$ `each month` translates to the aggregation window function w, where $w(t) = 31$ if t corresponds to January 31 and $w(t) = 28$ if t corresponds to February 28. We let Ω_{ω_1} be the function denoted by ω_1 and the time granularity of R_1', ..., R_k'.

Every TQuel retrieve statement of the form of (10) is equivalent to an expression in the historical algebra of the form

$$R = \hat{\pi}_{N_{i_1,a_1},\ldots,N_{i_n,a_n},N_{agg_1,p}}(\hat{\delta}_{\Gamma_\tau,\mathbf{Extend}(\Phi_\upsilon,\mathbf{Pred}(\Phi_\chi))} \cap N_{\bar{j}_1,1} \cap \cdots \cap N_{\bar{j}_x,1} \cap N_{agg_1,p}(\hat{\sigma}_{\Psi_\psi \wedge N_{j_2,c_2}=N_{agg_1,1} \wedge \cdots \wedge N_{j_p,c_p}=N_{agg_1,p-1}}(\mathbf{T}(R_1')\hat{\times}\cdots\hat{\times}\mathbf{T}(R_k')\hat{\times}R_{agg_1}))) \quad (11)$$

where

$$R_{agg_1} = \widehat{A}_{f_1,\Omega_{\omega_1},N_{j_1,c_1},\{N_{j_2,c_2},\ldots,N_{j_p,c_p}\}}(\hat{\delta}_{\Gamma_{\tau_1},N_{\bar{j}_1,1},\ldots,N_{\bar{j}_x,m_{\bar{j}_x}}}(\hat{\sigma}_{\Psi_{\psi_1}}(\mathbf{T}(R'_{\bar{j}_1})\hat{\times}\cdots\hat{\times}\mathbf{T}(R'_{\bar{j}_x}))), \hat{\pi}_{N_{j_1,c_1},\ldots,N_{j_p,c_p}}(\mathbf{T}(R'_{\bar{j}_1})\hat{\times}\cdots\hat{\times}\mathbf{T}(R'_{\bar{j}_x}))) \quad (12)$$

over the scheme $\mathcal{N}_{agg_1} = \{N_{agg_1,1}, \ldots, N_{agg_1,p}\}$, where $\forall u, 1 \le u \le p-1$, $N_{agg_1,u} = N_{j_{u+1},c_{u+1}}$ and $N_{agg_1,p}$ is the attribute name associated with the aggregate value. Here we assume that f_1 is the family of scalar aggregates (e.g., COUNTINT) corresponding to the family of TQuel aggregates f_1' (e.g., `count`). Expression (12) applies the where and when predicates to the cartesian product of the relations associated with tuples variables appearing in the aggregate, and applies the aggregate operator to the result. Expression (11) differs only slightly from the expression (3) on page 30 for a retrieve statement without aggregates. The expanded selection operator provides the necessary linkage between the attributes in the aggregate's by-list and corresponding attributes in the base relations. The expanded derivation operator imposes the TQuel restriction that the valid time of tuples in the derived relation be the intersection of the valid time specified in the valid clause, the valid times of the tuples in the base relations participating in the aggregation, and the valid time of the aggregate itself. Of course, if f_1' is a unique aggregate, then $\widehat{AU}$ should be used instead of $\widehat{A}$ in (12).

Two changes to (11) are required to handle special cases. First, if a tuple variable $\bar{j}_u$, $1 \le u \le x$, does not appear outside the aggregate f_1' in (10), then $N_{\bar{j}_u,1}$ does not appear in the second subscript

$$\mathcal{N}_1 = \{N_{1,1},\ \ldots,\ N_{1,m_1},\ \mathit{From}_1,\ \mathit{To}_1\}$$
$$\ldots$$
$$\mathcal{N}_k = \{N_{k,1},\ \ldots,\ N_{k,m_k},\ \mathit{From}_k,\ \mathit{To}_k\}$$

where, for notational convenience, we assume that $N_{1\,1},\ \ldots,\ N_{k,m_k}$ are unique. Also, let

$i_1,\ i_2,\ \ldots,\ i_n$ and $j_1,\ j_2,\ \ldots,\ j_p$ be integers, not necessarily distinct, in the range 1 to k, indicating the tuple variables (possibly repeated) appearing in the target list and aggregate, respectively;

$a_l,\ 1 \le l \le n$, be an integer in the range 1 to m_{i_l}, indicating the attribute names appearing in the target list where $(\forall u)(\forall v),\ (1 \le u \le n \wedge 1 \le v \le n \wedge u \ne v \wedge i_u = i_v),\ a_u \ne a_v$;

$c_h,\ 1 \le h \le p$, be an integer in the range 1 to m_{j_h}, indicating the attribute names appearing in the aggregate where $(\forall u)(\forall v),\ (1 \le u \le p \wedge 1 \le v \le p \wedge u \ne v \wedge j_u = j_v),\ c_u \ne c_v$; and

$\bar{j}_1,\ \bar{j}_2,\ \ldots,\ \bar{j}_x$ be the distinct integers in $j_1,\ j_2,\ \ldots,\ j_p$ where $\bar{j}_1 = j_1$, indicating the x (non-repeated) tuple variables appearing in the aggregate.

Then, the TQuel retrieve statement with the aggregate f'_1 in the target list has the following syntax

```
range of r'_1 is R'_1
...
range of r'_k is R'_k
retrieve into R'_{k+1}(N_{k+1,1} = r'_{i_1}.N_{i_1,a_1}, ..., N_{k+1,n} = r'_{i_n}.N_{i_n,a_n},
                       N_{k+1,n+1} = f'_1(r'_{j_1}.N_{j_1,c_1} by r'_{j_2}.N_{j_2,c_2}, ..., r'_{j_p}.N_{j_p,c_p}
                                          for ω_1
                                          where ψ_1
                                          when τ_1))                    (10)
    valid from υ to χ
    where ψ
    when τ
```

This statement computes a new relation R'_{k+1} over the relational scheme

$$\mathcal{N}_{k+1} = \{N_{k+1,1},\ \ldots,\ N_{k+1,n},\ N_{k+1,n+1},\ \mathit{From}_{k+1},\ \mathit{To}_{k+1}\}$$

over the scheme $\mathcal{N}_{position} = \{N_{position}\}$.

EXAMPLE. If we assume an aggregation window function $w(t) = 0$ and an empty set of by-clause attributes, then `earliest` for attribute *State* of relation S_3 is

$$\hat{\sigma}_{N_{earliest,1}=N_{earliest,2}}(\widehat{A}_{\text{SMALLEST},\,0,\,N_{position},\,\emptyset}(R_{position},\ R_{position})\ \hat{\times}\ R_{position}) =$$

$$\Big\{\ \langle(1,\ \{1,2\}),\ (1,\ \{1,2\})\rangle,$$
$$\langle(2,\ \{3\}),\ (2,\ \{1,2,3\})\rangle,$$
$$\langle(3,\ \{4,5,6\}),\ (3,\ \{4,5,6\})\rangle,$$
$$\langle(5,\ \{7,8\}),\ (5,\ \{7,8\})\rangle\ \Big\}$$

where $R_{position} = \hat{\sigma}_{N_{position} \neq 0}(\widehat{A}_{\text{POSITION},\,\infty,\,State,\,\emptyset}(\mathrm{S}_3,\ \mathrm{S}_3)) =$

$$\Big\{\ \langle(1,\ \{1,2\})\rangle,$$
$$\langle(2,\ \{1,2,3\})\rangle,$$
$$\langle(3,\ \{4,5,6\})\rangle,$$
$$\langle(4,\ \{5,6\})\rangle,$$
$$\langle(5,\ \{7,8\})\rangle\ \Big\}$$ □

The interval $\{1,2\}$ results at times 1 and 2; the interval $\{1,2,3\}$ results at time 3; the interval $\{4,5,6\}$ results at times 4, 5, and 6; and the interval $\{7,8\}$ results at times 7 and 8. The resulting value components can be effectively ignored.

As illustrated in this example, the algebraic equivalent of `earliest` is a two-attribute historical relation. The valid component of the first attribute is the time when the valid component of the second attribute was the earliest interval. Also note that the value component of both attributes is the position of the valid component of the second attribute in $\mathbf{ORDERINT}_N(R)$.

4.3.1 TQuel Aggregates in the Target List

In Section 4.2 we showed the algebraic equivalent of the TQuel retrieve statement without aggregates. We now show the algebraic equivalent of a TQuel retrieve statement with aggregates in its target list. We consider changes to the algebraic expression to support one non-unique aggregate in the target list only; similar changes would be needed for each additional aggregate in the target list.

Once again assume that we are given the k snapshot relations $R'_1, \ldots, R'_k$ whose schemes are respectively,

The first clause states that each interval in the time-stamp of attribute N of a tuple in R appears in S, the second clause states that no additional intervals are present, and the third clause provides the ordering conditions.

Now, we can define a family of scalar aggregate functions POSITION_{N_a}, $1 \leq a \leq m$, where POSITION_N first identifies, for a tuple q and time t, the interval in the valid component of attribute N in q that overlaps t and then calculates the position of that interval in $\mathbf{ORDERINT}_N(R)$, for an historical relation R. If no interval in the valid component of attribute N overlaps t or the interval is not in $\mathbf{ORDERINT}_N(R)$, POSITION_N returns zero.

$$\text{POSITION}_N(q,\ t,\ R) = \begin{cases} v,\ S[v] = I & \text{if } \exists I \in \mathbf{Interval}(valid(q_N)) \\ \quad \wedge\ 1 \leq v \leq |\mathbf{ORDERINT}_N(R)| & \\ \quad \wedge\ S[v] \in \mathbf{ORDERINT}_N(R) & \\ \quad \wedge\ t \in S[v] & \\ 0 & \text{Otherwise} \end{cases}$$

Note that POSITION, unlike COUNTINT and FIRSTVALUE, requires parameters q and t, as well as R.

Now assume that we are given a family of scalar aggregate functions SMALLEST_{N_a}, $1 \leq a \leq m$, where SMALLEST_N produces the smallest value of numeric attribute N. That is,

$$\text{SMALLEST}_N(q,\ t,\ R) = \begin{cases} value(r_N), & \text{if } R \neq 0 \\ \quad \forall r' \in R\ (value(r_N) \leq value(r'(N))) & \\ 0 & \text{Otherwise} \end{cases}$$

The families of scalar aggregates POSITION and SMALLEST are both needed to define the algebraic equivalent of the TQuel aggregate `earliest` for attribute N of relation R'. First, POSITION is used to assign each interval in the time-stamp of attribute N of a tuple in $\mathbf{T}(R')$ to an integer representing the interval's relative position in the chronological ordering of intervals. Then, SMALLEST is used to determine, from this assignment of intervals to integers, the times, if any, when each interval was the earliest interval. We encode the required information in the valid component of the attributes. The second attribute will indicate the interval returned by the aggregate, and the first attribute will indicate when that interval is the correct result of earliest. The valid components will be used only during processing of the aggregate. If we assume an aggregation window function $w(t) = 0$ and an empty set of by-clause attributes, the algebraic equivalent of the TQuel aggregate `earliest` for attribute N of relation R' is

$$\hat{\sigma}_{N_{earliest,1}=N_{earliest,2}}(\widehat{A}_{\text{SMALLEST},\ 0,\ N_{position},\ \emptyset}(R_{position},\ R_{position}) \hat{\times} R_{position}) \tag{8}$$

over the scheme $\mathcal{N}_{earliest} = \{N_{earliest,1},\ N_{earliest,2}\}$ where

$$R_{position} = \hat{\sigma}_{N_{position} \neq 0}(\widehat{A}_{\text{POSITION},\ \infty,\ N_a,\ \emptyset}(R,\ R)) \tag{9}$$

function. They *were* used, however, in the computation of the subset of the original relation(s) passed to the function as the value of the third parameter, R. These first two parameters are used for the temporal constructor aggregates `earliest` and `latest`.

Next, we consider the TQuel aggregate `first`, which computes the oldest value. This aggregate requires a family of scalar aggregate functions FIRSTVALUE$_{N_a}$, $1 \leq a \leq m$.

$$\begin{aligned}\text{FIRSTVALUE}_N(q,\ t,\ R) \in \{u \mid\ & R \neq \emptyset \rightarrow \exists r \in R\ (\ \ \forall r' \in R\ (\mathbf{First}(r_N) \leq \mathbf{First}(r'_N)) \\ & \qquad\qquad \wedge\ u = \mathit{value}(r_N) \\ & \qquad\qquad)\\ & \wedge\ R = \emptyset \rightarrow u = \mathbf{NULLVALUE}(N) \\ \}\end{aligned}$$

where **NULLVALUE** is an auxiliary function that returns a special null value for the domain associated with its argument. Note that the set $\{u \mid \ldots\}$ need not be a singleton set. If there are two or more elements in the set, FIRSTVALUE returns only one element, that element being selected arbitrarily. This procedure is the same as that used by the TQuel aggregate `first` to select the oldest value of an attribute when there are multiple values that satisfy the selection criteria. If R is empty, FIRSTVALUE returns a special null value for the domain associated with attribute N.

Finally, we define the algebraic equivalent of the TQuel aggregate `earliest`, which computes the interval with the earliest starting time-stamp. `earliest` may be used in the when or valid clauses of TQuel. Unlike other TQuel aggregates, which produce a distribution of scalar values over time, `earliest` produces a distribution of intervals over time. Defining an algebraic equivalent for this aggregate is slightly more complicated owing to this distinction. We first introduce a family of auxiliary functions **ORDERINT**$_{N_a}$, $1 \leq a \leq m$, which orders chronologically all distinguishable intervals in the time-stamp of attribute N_a for tuples of historical relation R. Evaluating **ORDERINT**$_N(R)$ results in a sequence of the intervals appearing in the time-stamp of attribute N of tuples in R. The intervals are ordered from earliest starting time to latest starting time. When two or more intervals have the same starting time, they are ordered from the earliest stopping time to the latest stopping time. Let S denote a sequence of length $|S|$ and $S[v]$ denote the vth element of S.

$$\begin{aligned}S \triangleq\ \mathbf{ORDERINT}_N(R) \leftrightarrow\ & \forall r \in R\ \forall I \in \mathbf{Interval}(\mathit{valid}(r_N))\ \exists v,\ 1 \leq v \leq |S|\ (S[v] = I) \\ & \wedge\ \forall v,\ 1 \leq v \leq |S|\ \exists r \in R\ \exists I \in \mathbf{Interval}(\mathit{valid}(r_N))\ (S[v] = I)) \\ & \wedge\ \forall v,\ 2 \leq v \leq |S|\ (\\ & \qquad \mathbf{First}(S[v-1]) < \mathbf{First}(S[v]) \\ & \qquad \vee\ (\mathbf{First}(S[v-1]) = \mathbf{FIRST}(S[v]) \\ & \qquad\ \wedge\ \mathbf{Last}(S[v-1]) < \mathbf{Last}(S[v])))\end{aligned}$$

significant difference. Because an historical relation represents the changing value of its attributes and aggregates are computed from the entire relation, aggregates in TQuel return a distribution of values over time. Hence, while in Quel an aggregate with no by-list returns a single value, in TQuel the same aggregate returns a sequence of values, each assigned its valid times. When there is a by-list, an aggregate in TQuel returns a sequence of values for each value of the attributes in the by-list.

Several aggregates are only unique to TQuel: standard deviation (`stdev` and `stdevU`), average time increment (`avgti`), the variability of time spacing (`varts`), oldest value (`first`), newest value (`last`), *From-To* interval with the earliest *From* time (`earliest`), and *From-To* interval with the latest *From* time (`latest`).

Each TQuel aggregate has a counterpart in the historical algebra. The algebraic equivalents of TQuel aggregates are defined in terms of the historical aggregate functions $\widehat{A}$ and $\widehat{AU}$, which were defined in Section 3.4.Before defining the algebraic equivalents of TQuel aggregates in the context of a TQuel retrieve statement however, we consider the families of scalar aggregates that appear as parameters to $\widehat{A}$ and $\widehat{AU}$ in the algebraic equivalents of TQuel aggregates. Each aggregate in one of these families of scalar aggregates returns, for a partition of historical relation R at time t, the same value returned by its analogous TQuel scalar aggregate for a partition of relation R' at time t, where $R = \mathbf{T}(R')$.

We define here the families of scalar aggregates that appear as parameters to $\widehat{A}$ and $\widehat{AU}$ in the algebraic equivalents of the TQuel aggregates `count`, `countU`, `first`, and `earliest`. The `count` and `countU` aggregates illustrate how conventional aggregate operators, now applied to historical relations, can be handled. The `first` aggregate is an example of an aggregate that evaluates to a non-temporal domain such as character but uses an attribute's valid time in a way different from the conventional aggregate operators. Finally, `earliest` illustrates an aggregate that evaluates to an interval.

For the definitions that follow, let R be an historical relation of m-tuples over the relation scheme $\mathcal{N} = \{N_1, \ldots, N_m\}$ and Q be an historical relation over an arbitrary subscheme of $\mathcal{N}$.

The scalar aggregate family COUNT, introduced on page 13, is sufficient only to define the algebraic equivalent of the TQuel aggregates `count` and `countU` for an aggregation window of length zero (i.e., an instantaneous aggregate). Hence, we define another family of scalar aggregates COUNTINT_{N_a}, $1 \leq a \leq m$, that accommodates aggregation windows of arbitrary length by counting intervals rather than values.

$$\text{COUNTINT}_N(q,\ t,\ R) = \sum_{r \in R} |\mathbf{Interval}(valid(r_A))|$$

where N is an attribute of both Q and R, $q \in Q$, and $t \in \mathcal{T}$. Recall that **Interval**, formally defined in Appendix B, returns the set of intervals contained in its argument. Hence, COUNTINT_N simply sums the number of intervals in the time-stamp of attribute N of each tuple in R. Note that the first two parameters to this function, q, the tuple providing the partitioning attributes, and t, the time at which the aggregate is being evaluated, are not used in the definition of the

$$
\begin{aligned}
\forall j,\ 1 \le j \le n\ \exists r'_1 \in R'_1\ \cdots\ \exists r'_k \in R'_k\ (& \\
& value(r(N_j)) = r'_{i_j}(N_{i_j,a_j}) \\
& \wedge\ \Psi'_{\psi}(r'_1(N_{1,1}),\ \ldots,\ r'_k(N_{k,m_k})) \qquad (6) \\
& \wedge\ \Gamma'_{\tau}((r'_1(From_1),\ r'_1(To_1)),\ \ldots,\ (r'_k(From_k),\ r'_k(To_k))) \\
& \wedge\ \forall t \in valid(r(N_j))\ (\\
& \qquad t \in \mathbf{EXTEND}(\Phi'_{\upsilon}((r'_1(From_1),\ r'_1(To_1)),\ \ldots,\ (r'_k(From_k),\ r'_k(To_k))), \\
& \qquad\qquad \mathbf{PRED}(\Phi'_{\chi}((r'_1(From_1),\ r'_1(To_1)),\ \ldots,\ (r'_k(From_k),\ r'_k(To_k))))) \\
& \ ))) \\
&)
\end{aligned}
$$

This is P_3 in the proof outline.

The third clause of (4) on page 32 (i.e., the **NotNull** predicate) implies that

$$\forall r \in R_{k+1},\ \exists c\ 1 \le c \le n\ \exists t,\ t \in valid(r(N_c))$$

Applying the tuple calculus statement for R'_{k+1} in (1) on page 26 to (6) results in

$$
\begin{aligned}
\exists r' \in R'_{k+1}(& \quad \forall l,\ 1 \le l \le n\ (value(r(N_l)) = r'(N_{k+1,l})) \\
& \wedge\ \forall l,\ 1 \le l \le n\ \forall t \in valid(r(N_l))\ (t \in \mathbf{EXTEND}(r'(From),\ \mathbf{PRED}(r'(To)))) \\
&)
\end{aligned}
$$

This is P_5 in the proof outline.

Thus, the first clause in the definition of $\mathbf{T}(R'_{k+1})$ is shown to hold. A similar argument can be made, starting with the second main clause of (4), to show that the second clause of $\mathbf{T}$ holds, thereby demonstrating that R_{k+1} and R'_{k+1} are equivalent, and hence the historical algebra expression is equivalent to the indicated TQuel retrieve statement. ∎

So far we have dealt only with the core of TQuel, the retrieve statement without aggregates. The next two sections consider the full language, examining aggregates in some detail and the other statements and transaction time briefly.

4.3 TQuel Aggregates

TQuel aggregates [Snodgrass et al. 1989] are a superset of the Quel aggregates. Hence, each of Quel's six non-unique aggregates (i.e., `count`, `any`, `sum`, `avg`, `min`, and `max`) and three unique aggregates (i.e., `countU`, `sumU`, and `avgU`) has a TQuel counterpart. The TQuel version of each of these aggregates performs the same fundamental operation as its Quel counterpart, with one

$$
\begin{array}{ll}
10 & \quad \wedge\, \forall r_1 \in \mathbf{T}(R'_1)\ \cdots\ \forall r_k \in \mathbf{T}(R'_k) \qquad (4)\\
11 & \qquad\quad \forall I_1 \in \mathbf{INTERVAL}(\mathit{valid}(r_1(N_{1,1})))\ \cdots\\
12 & \qquad\qquad \forall I_k \in \mathbf{INTERVAL}(\mathit{valid}(r_k(N_{k,1})))\ (\\
13 & \qquad\qquad\quad (\ \forall l,\ 1 \le l \le n\ (\mathit{value}(r_{i_l}(N_{i_l,a_l})) = \mathit{value}(r(N_l)))\\
14 & \qquad\qquad\quad \wedge\ \Psi_\psi(\mathit{value}(r_1(N_{1,1})),\ \ldots,\ \mathit{value}(r_k(N_{k,m_k})))\\
15 & \qquad\qquad\quad \wedge\ \Gamma_\tau(I_1,\ \ldots,\ I_k))\\
16 & \qquad\qquad\quad \Rightarrow\ \forall j,\ 1 \le j \le n\ (\\
17 & \qquad\qquad\qquad \mathbf{EXTEND}(\Phi_\upsilon(I_1,\ \ldots,\ I_k),\ \mathbf{PRED}(\Phi_\chi(I_1,\ \ldots,\ I_k))) \subseteq \mathit{valid}(r(N_j))))\\
18 & \quad \wedge\, \exists l,\ 1 \le l \le n\ (\mathit{valid}(r(N_l)) \neq \emptyset)\\
19 & \}
\end{array}
$$

The first two main clauses (lines 1–9 and lines 10–17) in the above calculus statement correspond to the two clauses in the definition of **Reduce**. The $\hat{\times}$ operator contributes the phrase $r_1 \in \mathbf{T}(R'_1)\ \cdots\ r_k \in \mathbf{T}(R'_k)$ that appears in lines 1 and 10 of the calculus statement. The $\hat{\sigma}$ operator contributes the predicate found on lines 5 and 14 and the $\hat{\delta}$ operator contributes the predicates found on lines 6–8, and 15–17. The $\hat{\pi}$ operator contributes lines 4 and 13 and the **NotNull** predicate, line 18. Lines 1–18 correspond to the P_1 predicate in the proof outline.

We now use the definitions and lemmas presented earlier, along with set theory, to reduce the tuple calculus for R_{k+1} to $\mathbf{T}(R'_{k+1})$. The first clause in (4), along with Lemma 2, implies that

$$
\begin{array}{l}
\forall j,\ 1 \le j \le n\ \exists r'_1 \in R'_1\ \cdots\ \exists r'_k \in R'_k\ (\\
\qquad \mathit{value}(r(N_j)) = r'_{i_j}(N_{i_j,a_j})\\
\qquad \wedge\ \Psi_\psi(r'_1(N_{1,1}),\ \ldots,\ r'_k(N_{k,m_k})) \qquad (5)\\
\qquad \wedge\ \Gamma_\tau(\mathbf{EXTEND}(r'_1(\mathit{From}_1),\ \mathbf{PRED}(r'_1(\mathit{To}_1))),\ \ldots,\\
\qquad\qquad \mathbf{EXTEND}(r'_k(\mathit{From}_k),\ \mathbf{PRED}(r'_k(\mathit{To}_k))))\\
\qquad \wedge\ \forall t \in \mathit{valid}(r(N_j))\ (\\
\qquad\qquad t \in \mathbf{EXTEND}(\Phi_\upsilon(\mathbf{EXTEND}(r'_1(\mathit{From}_1),\ \mathbf{PRED}(r'_1(\mathit{To}_1))),\ \ldots,\\
\qquad\qquad\qquad\qquad \mathbf{EXTEND}(r'_k(\mathit{From}_k),\ \mathbf{PRED}(r'_k(\mathit{To}_k)))),\\
\qquad\qquad\qquad\qquad \mathbf{PRED}(\Phi_\chi(\mathbf{EXTEND}(r'_1(\mathit{From}_1),\ \mathbf{PRED}(r'_1(\mathit{To}_1))),\ \ldots,\\
\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{EXTEND}(r'_k(\mathit{From}_k),\ \mathbf{PRED}(r'_k(\mathit{To}_k)))))\\
\qquad))))
\end{array}
$$

This is P_2 in the proof outline.

Applying Lemma 1 to (5) results in

$$R_{k+1} = \{r \mid P_1(r, \mathbf{T}(R'_1), \ldots, \mathbf{T}(R'_k), \Psi_\psi, \Gamma_\tau, \Phi_\upsilon, \Phi_\chi)\}$$

We successively transform the predicate. Applying the definition of $\mathbf{T}$ yields a predicate over R'_i rather than $\mathbf{T}(R_i)$.

$$P_2(r, R'_1, \ldots, R'_k, \Psi_\psi, \Gamma_\tau, \Phi_\upsilon, \Phi_\chi)$$

The syntactic to semantic mappings Ψ_ψ, Γ_τ, Φ_υ, and Φ_χ, which assume attribute-value time-stamps, are analogous to the mappings Ψ'_ψ, Γ'_τ, Φ'_υ, and Φ'_χ, which assume tuple time-stamps. Exploit this similarity to yield a predicate over the primed versions of the mappings.

$$P_3(r, R'_1, \ldots, R'_k, \Psi'_\psi, \Gamma'_\tau, \Phi'_\upsilon, \Phi'_\chi)$$

The tuple calculus semantics for TQuel for the query (1) yields the following, which is the predicate in (2).

$$r \in R'_{k+1} \leftrightarrow P_4(r, R'_1, \ldots, R'_k, \Psi'_\psi, \Gamma'_\tau, \Phi'_\upsilon, \Phi'_\chi)$$

Applying this to P_3 yields a predicate over r and R'_{k+1}.

$$P_5(r, R'_{k+1})$$

which is then shown to imply the predicate in $\mathbf{T}$. Hence

$$R_{k+1} = \mathbf{T}(R'_{k+1})$$

We now go down a level of detail and specify each of the predicates present in the proof outline.

The tuple calculus statement for R_{k+1} is as follows.

$$R_{k+1} = \hat{\pi}_{N_{i_1,a_1},\ldots,N_{i_n,a_n}}(\hat{\delta}_{\Gamma_\tau, \mathbf{EXTEND}(\Phi_\upsilon, \mathbf{PRED}(\Phi_\chi))}(\hat{\sigma}_{\Psi_\psi}(\mathbf{T}(R'_1)\hat{\times}\cdots\hat{\times}\mathbf{T}(R'_k)))) \stackrel{\Delta}{=}$$

$$\begin{aligned}
&\{r^n \mid \forall j, 1 \le j \le n\ (\exists r_1 \in \mathbf{T}(R'_1)\ \cdots\ \exists r_k \in \mathbf{T}(R'_k)\\
&\qquad \exists I_1 \in \mathbf{INTERVAL}(\mathit{valid}(r_1(N_{1,1})))\cdots\\
&\qquad\quad \exists I_k \in \mathbf{INTERVAL}(\mathit{valid}(r_k(N_{k,1})))\ (\\
&\qquad\qquad \forall l,\ 1 \le l \le n,\ \mathit{value}(r(N_l)) = \mathit{value}(r_{i_l}(N_{i_l,a_l})))\\
&\qquad\qquad \wedge\ \Psi_\psi(\mathit{value}(r_1(N_{1,1})), \ldots, \mathit{value}(r_k(N_{k,m_k})))\\
&\qquad\qquad \wedge\ \Gamma_\tau(I_1, \ldots, I_k)\\
&\qquad\qquad \wedge\ \forall t \in \mathit{valid}(r(N_j))\\
&\qquad\qquad\qquad (t \in \mathbf{EXTEND}(\Phi_\upsilon(I_1, \ldots, I_k), \mathbf{PRED}(\Phi_\chi(I_1, \ldots, I_k))))\\
&\qquad\qquad)
\end{aligned}$$

represents cartesian product of the relations associated with tuple variables, followed by selection by the where-clause predicate, and then projection on the attributes in the target list. Similarly, every TQuel retrieve statement is equivalent to an algebraic expression that represents cartesian product of the referenced relations, followed by selection by the where-clause predicate, historical derivation as specified by the when and valid clauses, and then projection on the attributes in the target list. In particular, the derivation operator applies the predicate Γ_τ to each tuple, evaluates Φ_υ and Φ_χ to get the starting and ending events for the interval, and then constructs an interval bounded by these events.

EXAMPLE. List the state of residence for the periods when each student was enrolled.

```
range of C is Courses
range of H is Home
retrieve into HomeWhenEnrolled (Name = H.Name, State = H.State)
         valid from begin of (C overlap H) to end of (C overlap H)
         where C.Name = H.Name
         when C overlap H
```

This translates into the *HomeWhenEnrolled* relation given on page 8, with the following definition.

$$\hat{\pi}_{Home.Name,\ Home.State}(\hat{\delta}_{\Gamma,\ \mathbf{EXTEND}(\Phi_\upsilon, \mathbf{PRED}(\Phi_\chi))}(\hat{\sigma}_{Courses.Name=Home.Name}(Courses\ \hat{\times}\ Home))$$

where $\Gamma \equiv (\mathbf{Last}(\mathbf{First}(SName), \mathbf{First}(State)) < \mathbf{First}(\mathbf{Last}(SName), \mathbf{Last}(State)))$

$\Phi_\upsilon \equiv (\mathbf{Last}(\mathbf{First}(SName), \mathbf{First}(State)))$

$\Phi_\chi \equiv (\mathbf{First}(\mathbf{Last}(SName), \mathbf{Last}(State)))$ □

Theorem 4 *Every TQuel retrieve statement of the form of* (1) *found on page 26 is equivalent to an expression in the historical algebra of the form*

$$R_{k+1} = \hat{\pi}_{N_{i_1,a_1},\ ...,\ N_{i_n,a_n}}(\hat{\delta}_{\Gamma_\tau,\ \mathbf{Extend}(\Phi_\upsilon, \mathbf{Pred}(\Phi_\chi))}(\hat{\sigma}_{\Psi_\psi}(\mathbf{T}(R'_1)\hat{\times} \ldots \hat{\times}\mathbf{T}(R'_k)))). \qquad (3)$$

PROOF. To prove that R_{k+1} and R'_{k+1} are equivalent, we must show that $R_{k+1} = \mathbf{T}(R'_{k+1})$. We provide an overview of the proof, then delve into the details.

First, construct the tuple calculus statement for R_{k+1} from the definitions of the historical operators $\hat{\times}$, $\hat{\sigma}$, $\hat{\delta}$, and $\hat{\pi}$, using straightforward substitution, change of variable, and simplification (i.e., the definition of $\mathbf{T}(R'_1)\hat{\times} \cdots \hat{\times}\mathbf{T}(R'_k)$ obtained from the $\hat{\times}$ operator is substituted for references to the historical relation in the definition of $\hat{\sigma}$, etc.). The resulting statement includes a predicate over the input relations and the semantic equivalents of the various TQuel clauses.

Also let Γ_τ be the denotation in the historical algebra of τ. Γ_τ is obtained by replacing each occurrence of an ordered pair $(r'_i(From_i),\ r'_i(To_i))$ and each occurrence of a TQuel function in Γ'_τ with its algebraic equivalent according to the rules above and each occurrence of the predicates *precede*, *overlay*, and *equal* with its algebraic equivalent. That is,

$$
\begin{aligned}
I_1 \texttt{ precede } I_2 &\rightarrow \mathbf{Last}(I_1) < \mathbf{First}(I_2) \ \vee\ \mathbf{Last}(I_1) = \mathbf{First}(I_2), \\
I_1 \texttt{ overlap } I_2 &\rightarrow I_1 \cap I_2 \neq \emptyset, \text{ and} \\
I_1 \texttt{ equal } I_2 &\rightarrow I_1 = I_2.
\end{aligned}
$$

Note from the definition of $\mathbf{T}(R')$ that a tuple in $\mathbf{T}(R')$ has the same time-stamp for each of its attributes. Hence, although we require that each occurrences of an ordered pair $(r'_i(From_i),\ r'_i(To_i))$ in Φ'_υ, Φ'_χ, and Γ'_τ be replaced with the same attribute name (i.e., $N_{i,1}$), we could have specified any attribute of relation R_i.

We will need the following two lemmas in the equivalence proof to be presented shortly. The first relates the semantic functions used to formalize TQuel and those used in the historical algebra.

Lemma 1 *Φ_υ, Φ_χ, and Γ_τ are semantically equivalent to Φ'_υ, Φ'_χ, and Γ'_τ respectively. That is, the result of evaluating Φ'_υ, Φ'_χ, and Γ'_τ for tuples $r'_i \in R'_i$, $1 \leq i \leq k$, is the same as the result of evaluating Φ_υ, Φ_χ, and Γ_τ for the intervals I_i, $I_i = \mathbf{Extend}(r'_i(From_i),\ \mathbf{Pred}(r'_i(To_i)))$ substituted for the attribute name $N_{i,1}$.*

PROOF. The semantic equivalence follows directly from the definitions of the functions used in Φ'_υ, Φ'_χ, and Γ'_τ [Snodgrass 1987]. ∎

Finally, because TQuel assumes that value-equivalent tuples are coalesced, the valid times assigned value-equivalent tuples in R' are disjoint, non-adjacent intervals. Hence, the following lemma holds.

Lemma 2 *Each distinguishable interval of time in the attribute-value time-stamps in $\mathbf{T}(R')$ is in a time-stamp of each tuple in R'. That is,*

$$
\begin{aligned}
\forall r \in \mathbf{T}(R')\ \forall a,\ 1 \leq a \leq m\ \forall I \in \mathbf{Interval}(valid(r(N_a)))\ \exists r' \in R'(& \\
& \forall c,\ 1 \leq c \leq m\ (value(r(N_c)) = r'(N_c)) \\
& \wedge\ I = \mathbf{Extend}(r'(From),\ \mathbf{Pred}(r'(To))) \\
&)
\end{aligned}
$$

PROOF. Apply the definitions of coalescing and **Interval** to **T** and simplify. ∎

Having defined the algebraic equivalents of TQuel relations and expressions in the new TQuel clauses, we can now define the algebraic equivalent of a TQuel retrieve statement. Every Quel retrieve statement (a target list and where clause) is equivalent to an algebraic expression that

$$
\begin{aligned}
\mathbf{T}(R') \triangleq \{u^m \mid \quad & \exists r' \in R' (\quad \forall A \in \mathcal{N} (value(u(N_A)) = r'(N_A)) \\
& \qquad \wedge \forall A \in \mathcal{N} \, \forall t \in valid(u(N_A)) \, (t \in \mathbf{Extend}(r'(From), \mathbf{Pred}(r'(To)))) \\
& \qquad) \\
& \wedge \forall r' \in R' (\quad \forall A \in \mathcal{N} (r'(N_A) = value(u(N_A))) \\
& \qquad \wedge \forall A \in \mathcal{N} (\mathbf{Extend}(r'(From), \mathbf{Pred}(r'(To))) \subseteq valid(u(N_A))) \\
& \qquad) \\
\} &
\end{aligned}
$$

The first clause of this definition ensures that each tuple in $\mathbf{T}(R')$ has at least one value-equivalent tuple in R. The second clause in the definition ensures that each subset of value-equivalent tuples in R is represented by a single tuple in $\mathbf{T}(R')$. Note also that the same time-stamp is assigned to each attribute of a tuple in $\mathbf{T}(R')$. This time-stamp is simply the union of the time-stamps of the tuple's value-equivalent tuples in R'.

EXAMPLE. If R′ = { (Phil, English, 1, 1),
(Phil, English, 3, 4),
(Norman, English, 1, 2),
(Norman, Calculus, 5, 6) }

then T(R′) = *Courses*, shown on page 3. □

Definition 2 *We define a $m+2$-tuple TQuel relation R' and a m-tuple relation R in the historical algebra to be* equivalent *if, and only if, $R = \mathbf{T}(R')$. In addition, we define a TQuel query and an expression in the historical algebra to be* equivalent *if, and only if, they evaluate to equivalent relations.*

Let Ψ_ψ, Φ_υ, and Φ_χ be the denotations in the historical algebra of ψ, υ, and χ respectively. Ψ_ψ is obtained by replacing each occurrence of $r'_i(N_{i,a})$, $1 \le i \le k$, $1 \le a \le m_i$, in Ψ'_ψ with $N_{i,a}$. Φ_υ and Φ_χ are obtained by replacing each occurrence of an ordered pair $(r'_i(From_i), r'_i(To_i))$, $1 \le i \le k$, in Φ'_υ and Φ'_χ with $N_{i,1}$ and each occurrence of a TQuel function with its algebraic equivalent. That is,

r'_i	$\rightarrow$	$N_{i,1}$,
`begin of` I	$\rightarrow$	**First**(I),
`end of` I	$\rightarrow$	**Last**(I),
I_1 `overlap` I_2	$\rightarrow$	$I_1 \cap I_2$, and
I_1 `extend` I_2	$\rightarrow$	**Extend**(**First**(I_1), **Last**(I_2)).

$r'_i.N_{i,a} \rightarrow r'_i(N_{i,a})$,
`and` $\rightarrow \wedge$,
`or` $\rightarrow \vee$, and
`not` $\rightarrow \neg$.

Φ'_υ and Φ'_χ are obtained by replacing each occurrence of a tuple variable r'_i in υ and χ with the ordered pair $(r'_i(From_i),\ r'_i(To_i))$ and each occurrence of a temporal constructor with a corresponding function. That is,

$r'_i \rightarrow (r'_i(From_i),\ r'_i(To_i))$
`begin of` $I \rightarrow beginof(I)$,
`end of` $I \rightarrow endof(I)$
I_1 `overlap` $I_2 \rightarrow overlap(I_1,\ I_2)$, and
I_1 `extend` $I_2 \rightarrow extend(I_1,\ I_2)$

where *beginof*, *endof*, *overlap*, and *extend* are functions over intervals. Formal definitions for these functions are presented elsewhere [Snodgrass 1987].

Γ'_τ is obtained by replacing each occurrence of a logical operator in τ with its corresponding logical predicate according to the rules given for its replacement in ψ, replacing each occurrence of a tuple variable or temporal constructor according to the rules given for their replacement in υ and χ, and replacing each occurrence of a temporal predicate operator with an analogous predicate on intervals. That is,

I_1 `precede` $I_2 \rightarrow precede(I_1,\ I_2)$,
I_1 `overlap` $I_2 \rightarrow overlap(I_1,\ I_2)$, and
I_1 `equal` $I_2 \rightarrow equal(I_1,\ I_2)$

where *precede*, *overlap*, and *equal* are predicates over intervals. Again, formal definitions for these predicates are presented elsewhere [Snodgrass 1987].

4.2 Correspondence with the Historical Algebra

To compare the expressive power of TQuel and the historical algebra presented in Section 3, we first relate relations in the two systems, then expressions in the new TQuel clauses, and finally the retrieve statement with algebraic expressions.

Definition 1 *The transformation function* **T** *maps a TQuel embedded snapshot relation over the scheme* $\{N_1,\ \ldots,\ N_m,\ From,\ To\}$ *into its equivalent historical relation, valid in the historical algebra over the scheme* $\mathcal{N} = \{N_1,\ \ldots,\ N_m\}$.

$$\mathcal{N}_1 = \{N_{1,1},\ \ldots,\ N_{1,m_1},\ \mathit{From}_1,\ \mathit{To}_1\}$$
$$\ldots$$
$$\mathcal{N}_k = \{N_{k,1},\ \ldots,\ N_{k,m_k},\ \mathit{From}_k,\ \mathit{To}_k\}$$

For notational convenience, we associate " ′ " with TQuel relations, tuple variables, and expressions to differentiate them from their counterparts in the historical algebra and assume that $N_{1,1},\ \ldots,\ N_{k,m_k}$ are unique. The TQuel retrieve statement has the following syntax.

```
range of r'_1 is R'_1
...
range of r'_k is R'_k
retrieve into R'_{k+1}(N_{k+1,1} = r'_{i_1}.N_{i_1,a_1}, ..., N_{k+1,n} = r'_{i_n}.N_{i_n,a_n})      (1)
      valid from υ to χ
      where ψ
      when τ
```

where $i_1,\ i_2,\ \ldots,\ i_n$ are integers, not necessarily distinct, in the range 1 to k. This statement computes a new relation R'_{k+1} over the relational scheme

$$\mathcal{N}_{k+1} = \{N_{k+1,1},\ \ldots,\ N_{k+1,n},\ \mathit{From}_{k+1},\ \mathit{To}_{k+1}\}$$

Its tuple calculus statement has the following form [Snodgrass 1987].

$$\begin{aligned}
R'_{k+1} = \{u^{n+2} \mid\ & \exists r'_1 \in R'_1 \cdots \exists r'_k \in R'_k\ (\\
& u(N_{k+1,1}) = r'_{i_1}(N_{i_1,a_1}) \wedge \cdots \wedge u(N_{k+1,n}) = r'_{i_n}(N_{i_n,a_n}) \\
& \wedge\ u(\mathit{From}_{k+1}) = \Phi'_{\upsilon}((r'_1(\mathit{From}_1),\ r'_1(\mathit{To}_1)), \ldots, (r'_k(\mathit{From}_k),\ r'_k(\mathit{To}_k))) \\
& \wedge\ u(\mathit{To}_{k+1}) = \Phi'_{\chi}((r'_1(\mathit{From}_1),\ r'_1(\mathit{To}_1)), \ldots, (r'_k(\mathit{From}_k),\ r'_k(\mathit{To}_k))) \qquad (2) \\
& \wedge\ \mathit{Before}(u(\mathit{From}_{k+1}),\ u(\mathit{To}_{k+1})) \\
& \wedge\ \Psi'_{\psi}(r'_1(N_{1,1}), \ldots, r'_k(N_{k,m_k})) \\
& \wedge\ \Gamma'_{\tau}((r'_1(\mathit{From}_1),\ r'_1(\mathit{To}_1)), \ldots, (r'_k(\mathit{From}_k),\ r'_k(\mathit{To}_k))) \\
&)
\end{aligned}$$

$\}$

where *Before* is the "$<$" predicate on integers, the ordered pair $(r'_i(\mathit{From}_i),\ r'_i(\mathit{To}_i))$, $1 \le i \le k$, represents the interval $[r'_i(\mathit{From}_i),\ r'_i(\mathit{To}_i))$, and Ψ'_{ψ}, Φ'_{υ}, Φ'_{χ}, and Γ'_{τ} are the denotations described below of ψ, υ, χ, and τ respectively.

Ψ'_{ψ} is obtained by replacing each occurrence of an attribute reference $r'_i.N_{i,a}$, $1 \le i \le k$, $1 \le a \le m_i$, in ψ with $r'_i(N_{i,a})$ and each occurrence of a logical operator with its corresponding logical predicate. That is,

- Snapshot ($\widehat{SN}$) and AT convert between snapshot and historical relations.

The snapshot rollback (ρ) and historical rollback ($\hat{\rho}$) operators, defined elsewhere [McKenzie & Snodgrass 1990], serve to generalize the algebra to handle temporal relations incorporating both valid and transaction time.

We showed that intersection ($\hat{\cap}$), quotient ($\hat{\div}$), natural join ($\hat{\bowtie}$), and Θ-join ($\hat{\underset{\Theta}{\bowtie}}$) can all be defined in terms of the five basic operators, most in an identical fashion to the definition of their snapshot counterparts.

4 Equivalence with TQuel

We now show that the historical algebra defined above has the expressive power of the *TQuel* (*T*emporal *QUE*ry *L*anguage) [Snodgrass 1987] facilities that support valid time. TQuel is a version of Quel [Held et al. 1975], the calculus-based query language for the Ingres relational database management system [Stonebraker et al. 1976], augmented to handle both valid time and transaction time. Two new syntactic and semantic constructs are provided to support valid time. The *valid clause* is the temporal analogue to Quel's target list; it is used to specify the value of the valid time for tuples in the derived relation. This clause consists of the keywords `valid from to` and two temporal expressions, each consisting of tuple variables, temporal constants, and the temporal constructors `begin of`, `end of`, `overlap`, and `extend`. The *when clause* is the temporal analogue to Quel's where clause. This clause consists of the keyword `when` followed by a temporal predicate consisting of temporal expressions, the temporal predicate operators `precede`, `overlap`, and `equal`, and the logical operators `or`, `and`, and `not`. (Note that `overlap` is overloaded; it may be either a temporal constructor or a temporal predicate operator, with context differentiating the uses.) A third new construct, the *as-of* clause, is provided to handle transaction time but will not be considered here. We will generally limit our discussion of TQuel to its facilities for handling valid time.

Unlike the historical algebra, which assumes attribute-value time-stamping, TQuel assumes tuple time-stamping. The formal semantics of TQuel conceptually embeds its temporal relations in snapshot relations; such an embedding is done purely for convenience in developing the semantics. TQuel represents valid time by adding two time values to each tuple to specify the time when the tuple became valid (i.e., *From*) and the time when the tuple became invalid (i.e., *To*). Also unlike the historical algebra, TQuel allows value-equivalent tuples in a relation but assumes that value-equivalent tuples are *coalesced* (i.e., tuples with identical values for the explicit attributes neither overlap nor are adjacent in time). As we will see shortly, it is possible to convert the embedded, coalesced snapshot relations used in TQuel's formal semantics to historical relations.

4.1 TQuel Retrieve Statement

Assume that we are given the k snapshot relations $R'_1, \ldots, R'_k$ whose schemes are respectively,

Norman were included in $S_3 \hat{\div} S_6$ because neither resided in Texas at time 9. The $\hat{\delta}$ clause ensured that Norman was excluded even though he lived in Virginia at time 5 and in Texas at times 7 and 8. □

In defining the above operators, we restricted the valid-time component of attributes to elements from the domain $\wp(\mathcal{T})$. By so doing, we were able to define all operations on attribute time-stamps in terms of the standard operations from set theory. We can eliminate the restriction and allow time-stamps to have a more complex structure without difficulty. For example, we could allow the valid-time component of attributes to be an element from, or even a subset of, $\wp(\mathcal{T}) \times \wp(\mathcal{T})$. We need only redefine the functions *First*, *Last*, and *Interval* to handle time-stamps of the new form and replace each set operation on time-stamps with an equivalent operation for the new time domain. In this way, the algebra could support either periodicity [Lorentzos & Johnson 1988] or multi-dimensional time-stamps [Gadia & Yeung 1988].

We also restricted the value component of attributes to atomic elements from a value domain. Several of the other historical algebras that have been proposed allow set-valued attributes [Clifford & Croker 1987, Gadia 1986, Tansel 1986]. Their purpose in allowing set-valued attributes is to model real-world relationships more naturally and to eliminate the need to replicate data among tuples. These algebras only allow one level of nesting. Hence, while they can model the relationship between students and courses without replication of data, they can't model the relationships among students, courses, and grades without replication of data. Several proposals have already been presented for extending the snapshot algebra to support non-first-normal-form relations with an arbitrary level of nesting [Roth et al. 1988, Schek & Scholl 1986, Özsoyoğlu et al. 1987]. Hence, rather than complicate the semantics of the algebra by allowing set-valued attributes, we propose extending the algebra to support non-first-normal-form historical relations with an arbitrary level of nesting using an approach similar to the one Schek and Scholl used to extend the snapshot algebra. Then, we could define both relations and operations on relations recursively. At each recursively defined level, an attribute could take on an atomic value from a value domain or a structured value from a domain of historical relations. The semantics, however, would be left unchanged, simply embedded in the new structure.

We leave these last two extensions to future work.

3.7 Summary

We first introduced *historical relations*, in which attribute values are associated with set-valued time-stamps. We then defined ten historical operators.

- Five operators are analogous to the five standard snapshot operators: union ($\hat{\cup}$), difference ($\hat{-}$), cartesian product ($\hat{\times}$), selection ($\hat{\sigma}$), and projection ($\hat{\pi}$).
- Historical derivation ($\hat{\delta}$) effectively performs selection and projection on the valid-time dimension by replacing the time-stamp of each attribute of selected tuples with a new time-stamp.
- Aggregation ($\widehat{A}$) and unique aggregation ($\widehat{AU}$) serve to compute a distribution of single values over time for a collection of tuples.

$$S \hat{\div} R \triangleq \hat{\pi}_{\mathcal{A}_Q}(S) \hat{-} \hat{\pi}_{\mathcal{A}_Q}(U \hat{\cup} \hat{\delta}_{R_1 \neq \emptyset \vee \cdots \vee R_{m_2} \neq \emptyset, (Q_1, \mathcal{T}), \ldots, (Q_{m_1}, \mathcal{T}), (R_1, R_1), \ldots, (R_{m_2}, R_{m_2})}(U))$$

where $U = (\hat{\pi}_{\mathcal{A}_Q}(S) \hat{\times} R) \hat{-} S$. The additional restriction introduced by the $\hat{\delta}$ clause ensures that no tuple in $S \hat{\div} R$ can combine with a tuple in R to produces a tuple whose attribute time-stamps are not contained in the attribute time-stamps of its value-equivalent tuple in S.

EXAMPLES. Assume that we are given the historical relation S_3 from page 6, duplicated below.

$$\left\{ \begin{array}{l} \langle(\text{Phil, } \{1,3,4\}), (\text{Kansas, } \{1,2,3\})\rangle, \\ \langle(\text{Phil, } \{1,3,4\}), (\text{Virginia, } \{4,5,6\})\rangle, \\ \langle(\text{Norman, } \{1,2,5,6\}), (\text{Virginia, } \{1,2,5,6\})\rangle, \\ \langle(\text{Norman, } \{1,2,5,6\}), (\text{Texas, } \{7,8\})\rangle \end{array} \right\}$$

If we are given the following historical relation S_5,

$$\mathrm{S}_5 = \left\{ \begin{array}{l} \langle(\text{Virginia, } \{5\})\rangle, \\ \langle(\text{Texas, } \{7, 8\})\rangle \end{array} \right\}$$

then

$$\mathrm{S}_3 \hat{\div} \mathrm{S}_5 = \left\{ \langle(\text{Norman, } \{1, 2, 5, 6\})\rangle \right\}.$$

If, however, we are given

$$\mathrm{S}_6 = \left\{ \begin{array}{l} \langle(\text{Virginia, } \{5\})\rangle, \\ \langle(\text{Texas, } \{7, 8, 9\})\rangle \end{array} \right\}$$

then

$$\mathrm{S}_3 \hat{\div} \mathrm{S}_6 = \emptyset.$$

In the first example, although Phil lived in Virginia at time 5, he was not included in $\mathrm{S}_3 \hat{\div} \mathrm{S}_5$ because he did not reside in Texas at times 7 and 8. In the second example, neither Phil nor

$$Q \bowtie R \stackrel{\Delta}{=} \pi_{\mathcal{A}_Q, Q.I_1, ..., Q.I_m, \mathcal{A}_R}(\sigma_{Q.I_1 = R.I_1 \wedge \cdots \wedge Q.I_m = R.I_m}(Q \times R)).$$

Now let Q and R be historical relations, rather than snapshot relations. If we were to simply replace snapshot operators in the above definition with their historical counterparts, $\hat{\bowtie}$ would retain the valid time assigned to attributes $\mathcal{A}_{QR}$ in Q but not in R, because the projection somewhat arbitrarily keeps the common attributes from Q and not from R. Similarly, if we were also to replace references to Q_u, $1 \leq u \leq m$, in the projection operator with references to R_u, $\hat{\bowtie}$ would retain the valid time assigned to attributes $\mathcal{A}_{QR}$ in R but not in Q. Retention of the valid time assigned to attributes $\mathcal{A}_{QR}$ in both Q and R, however, seems more appropriate. Hence, we define $Q \hat{\bowtie} R$, the historical natural join of Q and R, as

$$Q \hat{\bowtie} R \stackrel{\Delta}{=} \hat{\pi}_{\mathcal{A}_Q, Q.I_1, ..., Q.I_m, \mathcal{A}_R}(\hat{\delta}_{true, Q_1, ..., Q_{m_1}, Q.I_1 \cup R.I_1, ..., Q.I_m \cup R.I_m, R_1, ..., R_{m_2}, R.I_1, ..., R.I_m}(\hat{\sigma}_{Q.I_1 = R.I_1 \wedge \cdots \wedge Q.I_m = R.I_m}(Q \hat{\times} R))).$$

The $\hat{\delta}$ operator is introduced to compute the valid-time component of attributes in the resulting historical relation common to both Q and R. Here, we use union semantics to retain, for each attribute common to Q and R, the valid time assigned to the attribute in both relation relations. We can just as easily define other historical variations of natural join using either intersection or difference semantics. Note that the new time-stamps for attributes R_1, ..., R_m are arbitrary as these attributes are discarded by the projection operator.

To define quotient, let S be a snapshot relation of $(m_1 + m_2)$-tuples on the relation signature Z_S with attributes $\mathcal{A}_Q \cup \mathcal{A}_R$ and R be a snapshot relation of m_2-tuples on the relation signature Z_R with attributes $\mathcal{A}_R$, where $\mathcal{A}_Q = \{ Q_1, \ldots, Q_{m_1} \}$ and $\mathcal{A}_R = \{ R_1, \ldots, R_{m_2} \}$. Then, the quotient of S divided by R ($S \div R$) intuitively is the maximal subset Q of $\pi_{\mathcal{A}_Q}(S)$ such that $Q \times R$ is contained in S [Maier 1983]. $S \div R$ is defined as [Ullman 1988]

$$S \div R \stackrel{\Delta}{=} \pi_{\mathcal{A}_Q}(S) - \pi_{\mathcal{A}_Q}((\pi_{\mathcal{A}_Q}(S) \times R) - S).$$

Now let S and R be historical relations, rather than snapshot relations. If we were to simply replace snapshot operators in the above definition with their historical counterparts, $\hat{\div}$ would not place the same restrictions on the attribute time-stamps of tuples in Q that it places on the tuples' attribute values. The operator would require that each tuple in $Q \hat{\times} R$ have a value-equivalent tuple in S, but it would not require that the attribute time-stamps of a tuple in $Q \hat{\times} R$ be contained in the attribute time-stamps of its value-equivalent tuple in S. Hence, we propose a definition of $\hat{\div}$ that places the same restrictions on the attribute time-stamps of tuples in Q that it places on the tuples' attribute values. If we let the historical quotient of S divided by R ($S \hat{\div} R$) be the maximal temporal contents of $\hat{\pi}_{\mathcal{A}_Q}(S)$ such that $Q \hat{\times} R$ is contained temporally in S, then $S \hat{\div} R$ is defined as

Historical intersection can be defined in an identical fashion to its snapshot counterpart. Definition of the historical version of intersection is straightforward only because we took care when defining the historical version of difference to ensure its compatible with definition of intersection in terms of difference. If we let Q and R be snapshot relations of m-tuples over the relation signature z with attributes $\mathcal{A} = \{ I_1, \ldots, I_m \}$, then $Q \cap R$ is defined as [Ullman 1988]

$$Q \cap R \triangleq Q - (Q - R).$$

Now, let Q and R be historical relations, rather than snapshot relations. Then, $Q \hat{\cap} R$, the historical intersection of Q and R, is defined as

$$Q \hat{\cap} R \triangleq Q \hat{-} (Q \hat{-} R).$$

Θ-join also can be defined in an identical fashion to its snapshot counterpart. Definition of the historical version of Θ-join is straightforward because its definition involves only selection and cartesian product, two operators whose historical versions are themselves defined in an identical fashion to their snapshot couterparts. If we let Q be a historical relation of m_1-tuples on the relation signature z_Q with attributes $\mathcal{A}_Q = \{ I_{Q,1}, \ldots, I_{Q,m_1} \}$ and R be a historical relation of m_2-tuples on the relation signature z_R with attributes $\mathcal{A}_R = \{ I_{R,1}, \ldots, I_{R,m_2} \}$, where $\mathcal{A}_Q \cap \mathcal{A}_R = \emptyset$, then the Θ-join of Q and R is defined as [Ullman 1988]

$$Q \underset{u\theta v}{\bowtie} R \triangleq \sigma_{u\theta v}(Q \times R),$$

where the u and v attributes are Θ-comparable.

Now let Q and R be historical relations, rather than snapshot relations. Then the historical Θ-join of Q and R is defined as

$$Q \underset{u\theta v}{\hat{\bowtie}} \triangleq \hat{\sigma}_{u\theta v} (Q \hat{\times} R).$$

Historical natural join and quotient, unlike historical difference and Θ-join, can't be defined simply by substituting historical operators for snapshot operators in the definition of their snapshot counterparts [Ullman 1988], because both involve projection, an operation whose semantics in the historical algebra is substantially different from its semantics in the snapshot algebra. Small, but important, changes must be made to the definitions to handle properly the temporal dimension. Let $\mathcal{A}_Q = \{ Q_1, \ldots, Q_{m_1} \}$, $\mathcal{A}_R = \{ R_1, \ldots, R_{m_2} \}$, and $\mathcal{A}_{QR} = \{ I_1, \ldots, I_m \}$. Also let Q be a snapshot relation of (m_1+m)-tuples on the relation signature Z_Q with attributes $\mathcal{A}_Q \cup \mathcal{A}_{QR}$ and R be a snapshot relation of (m_2+m)-tuples on the relation signature Z_R with attributes $\mathcal{A}_R \cup \mathcal{A}_{QR}$. Hence, the attributes $\mathcal{A}_{QR}$ are common to Q and R. Rather than rename attributes, we simply refer to the common attributes in Q and R as $Q.I_u$ and $R.I_u$, $1 \leq u \leq m$, respectively, for notational convenience. Then $Q \bowtie R$, the natural join of Q and R, is defined as [Ullman 1988]

historical relation(s). Because the historical algebra allows tuples that contain attributes of differing time-stamps, it satisfies this property only through the introduction of distinguished nulls when taking snapshots. We avoid this problem by proving a weaker property: we restrict reducibility to operations on historical relations that have identical time-stamps for all of their attributes, termed *homogeneous relations.*

Theorem 3 *The historical operators* $\hat{\cup}$, $\hat{-}$, $\hat{\times}$, $\hat{\sigma}$ *and* $\hat{\pi}$ *reduce to their snapshot counterparts when their arguments are homogeneous.*

PROOF. Let Q and R be homogeneous relations. We demonstrate reducibility for selection, specifically, for all values of τ,

$$\widehat{SN}(\hat{\sigma}_F(R),\ \tau) = \sigma_F(\widehat{SN}(R,\ \tau)).$$

$$\widehat{SN}(\hat{\sigma}_F(R),\ \tau) = \{(value(r_1),\ \ldots,\ value(r_n)) \mid \exists r \in \hat{\sigma}_F(R)\ \wedge\ \tau \in valid(r_1)\}$$

from the definition of $\widehat{SN}$ and the fact that R is homogeneous.

$$= \{(value(r_1),\ \ldots,\ value(r_n)) \mid \exists R \in r \wedge F(value(r_1),\ \ldots,\ value(r_n)) \wedge \tau \in valid(r_1)\}$$

from the definition of $\hat{\sigma}_F$.

$$= \{(r'_1,\ \ldots,\ r'_n) \mid \exists r \in R,\ \exists r'_1 \ldots \exists r'_n,\ r'_i = value(r_i) \wedge F(r'_1,\ \ldots,\ r'_n)) \wedge\ \tau \in valid(r_1)\}$$

by substituting r'_i for $value(r_i)$.

$$= \{(r'_1,\ \ldots,\ r'_n) \mid \exists r' \in \widehat{SN}(R,\ \tau)\ \wedge\ F(r'_1,\ \ldots,\ r'_n)\}$$

from the definition of $\widehat{SN}$.

$$= \sigma_F(\widehat{SN}(R,\ \tau))$$

from the definition of σ.

Proofs for the other four operators follow analogously. ∎

3.6 Additional Aspects of the Algebra

We have thus far defined eight algebraic operators. We now show how historical intersection ($\hat{\cap}$), Θ-join ($\overset{\hat{}}{\underset{\Theta}{\bowtie}}$), natural join ($\hat{\bowtie}$), and quotient ($\hat{\div}$) all can be defined in terms of the six operators $\hat{\cup}$, $\hat{-}$, $\hat{\times}$, $\hat{\sigma}$, $\hat{\pi}$, and $\hat{\delta}$.

A relational algebra is said to be *complete* if it is at least as expressive as the snapshot algebra [Codd 1972].

Theorem 2 *The historical algebra is complete.*

PROOF. We show this for selection, that is, we show that historical selection is as expressive as snapshot selection. Specifically, we show that for all values of τ,

$$\hat{\sigma}_F(AT(R', \tau)) = AT(\sigma_F(R'), \tau)$$

Here we use primes ($'$) to denote snapshot relations.

$$\hat{\sigma}_F(AT(R', \tau)) = \{u \mid \exists u \in AT(R', \tau) \wedge F(value(u_1), \ldots, value(u_m))\}$$

from the definition of $\hat{\sigma}$.

$$= \{u \mid \exists r' \in R' \, (\forall \, i, \, 1 \leq i \leq n \, (r'_i = value(u_i)) \wedge \; valid(u_i) = \tau)) \\ \wedge \, F(value(u_1), \ldots, value(u_m))\}$$

from the definition of AT.

$$= \{u \mid \exists r' \in R' \, (\forall \, i, \, 1 \leq i \leq n \, (r'_i = value(u_i) \wedge \; valid(u_i) = \tau) \wedge \; F(r'_1, \ldots, r'_m))\}$$

by substituting r'_i for $value(u_i)$.

$$= \{u \mid \exists r' \in \sigma_F(R') \, (\forall \, i, \, 1 \leq i \leq n \, (r'_i = value(u_i) \wedge \; valid(u_i) = \tau))\}$$

from the definition of σ.

$$= AT(\sigma_F(R'), \, \tau)$$

from the definition of AT.

The correspondence between historical and snapshot operators centers on how time-stamps are manipulated by the historical operators. Historical cartesian product doesn't alter the time-stamps. Historical projection merges the time-stamps for value equivalent tuples, which has no effect since all the time-stamps are $\{\tau\}$; the same holds for historical union. Finally, historical difference subtracts the chronons of value equivalent tuples, reducing all associated time-stamps to $\emptyset$, which removes the tuple. ∎

We now examine whether the historical algebra is in some sense a consistent extension of the snapshot algebra. An algebra is said to *reduce* to the snapshot algebra if taking a snapshot on the result of applying a historical operator on one or two historical relations is identical to the result of applying the analogous snapshot operator to the snapshots (at the same time) of the

$\widehat{SN}$ returns a snapshot relation, this criterion is not relevant. Finally, no two output tuples can be value-equivalent. This criterion is explicitly ensured in the semantics of $\hat{\pi}$ and $\hat{\delta}$ via the **Reduce** function. For the operators $\hat{\cup}$, $\hat{-}$, $\hat{\times}$ and $\hat{\sigma}$ we show that if there are value-equivalent tuples in an operator's output relation, then there must have been value-equivalent tuples in at least one of its input relations. For the operators $\widehat{A}$, and $\widehat{AU}$, we show by contradiction that there cannot be value-equivalent tuples in their output relations. Because $\widehat{SN}$ returns a snapshot relation, this criterion is not relevant.

Case 1: $\hat{\cup}$. Assume that $Q \,\hat{\cup}\, R$ contains at least two value-equivalent tuples. From the definition of $\hat{\cup}$, each tuple in $Q \,\hat{\cup}\, R$ has a value-equivalent tuple in Q, R, or both. If two value-equivalent tuples $\hat{u}_1$ and $\hat{u}_2$ in $Q \,\hat{\cup}\, R$ do not have a value-equivalent tuple in R, then both are tuples in Q. Similarly, if they do not have a value-equivalent tuple in Q, then both are tuples in R. If they have a value-equivalent tuple in both Q and R, then each was constructed from a value-equivalent tuple in Q and a value-equivalent tuple in R. If both $\hat{u}_1$ and $\hat{u}_2$ had been constructed from the same tuple in Q and the same tuple in R, then $\hat{u}_1$ and $\hat{u}_2$ would be, by definition, the same tuple. Hence, they were constructed from different value-equivalent tuples in Q, R, or both.

Case 2: $\hat{-}$. Assume that $Q \,\hat{-}\, R$ contains at least two value-equivalent tuples. From the definition of $\hat{-}$, each tuple in $Q \,\hat{-}\, R$ has a value-equivalent tuple in Q but not in R or a value-equivalent tuple in both Q and R. If two value-equivalent tuples $\hat{u}_1$ and $\hat{u}_2$ in $Q \,\hat{-}\, R$ do not have a value-equivalent tuple in R, then both are tuples in Q. If they have a value-equivalent tuple in both Q and R, then each was constructed from a value-equivalent tuple in Q and a value-equivalent tuple in R. If both $\hat{u}_1$ and $\hat{u}_2$ had been constructed from the same tuple in Q and the same tuple in R, then $\hat{u}_1$ and $\hat{u}_2$ would be, by definition, the same tuple. Hence, they were constructed from different value-equivalent tuples in Q, R, or both.

Case 3: $\hat{\times}$. Assume that $Q \,\hat{\times}\, R$ contains at least two value-equivalent tuples. From the definition of $\hat{\times}$, each tuple in $Q \,\hat{\times}\, R$ is constructed from a tuple in Q and a tuple in R. If two value-equivalent tuples $\hat{u}_1$ and $\hat{u}_2$ in $Q \,\hat{\times}\, R$ had been constructed from the same tuple in Q and the same tuple in R, then $\hat{u}_1$ and $\hat{u}_2$ would be, by definition, the same tuple. Hence, they were constructed from different value-equivalent tuples in Q, R, or both.

Case 4: $\hat{\sigma}$. Assume that $\hat{\sigma}_F(R)$ contains at least two value-equivalent tuples. From the definition of $\hat{\sigma}$, each tuple in $\hat{\sigma}_F(R)$ is a tuple in R. Hence, any two value-equivalent tuples in $\hat{\sigma}_F(R)$ are also tuples in R.

Case 5: AT. Assume that $AT(R,\ \tau)$ contains at least two value-equivalent tuples. Because AT generates at most one output tuple for each input tuple, there must have been two identical tuples in R, which is not possible, as R is a set.

Case 6: $\widehat{A}$. Assume that $\widehat{A}_{f,\,w,\,N_a,\,X}(Q,\ R)$ contains at least two value-equivalent tuples. From Case 1 above, if $\widehat{A}_{f,\,w,\,N_a,\,X}(Q,\ R)$ contains value-equivalent tuples, then the input relation to $\widehat{A}$'s outermost π operator, which explicitly has the **Reduce** function applied to it. Hence, our assumption that $\widehat{A}_{f,\,w,\,N_a,\,X}(Q,\ R)$ contains at least two value-equivalent tuples is contradicted.

Case 7: $\widehat{AU}$. Simply replace $\widehat{A}$ with $\widehat{AU}$ in Case 7. ∎

3.4.4 Expressions in Aggregates

The aggregate functions can be extended to allow expressions to be aggregated and support aggregation by arbitrary expressions. Let $Eaggregate$ be an arbitrary expression involving u historical aggregate functions. Also, assume that the vth historical aggregate function applies the scalar aggregate f_v to attribute N_{a_v} where the aggregation window function is w_v, and the partitioning attributes are X_v. Then the definition of $\widehat{A}$, now denoted by

$$\widehat{A}_{f_1,\ ...,\ f_u,\ w_1,\ ...,\ w_u,\ N_{a_1},\ ...,\ N_{a_u},\ X_1,\ ...,\ X_u,\ Eaggregate}(Q,\ R),$$

is constructed from the definition of $\widehat{A}$ above simply by substituting $y = Eaggregate'$ for $y = f_{N_a}(\ldots)$. $Eaggregate'$ is $Eaggregate$ where each reference to the vth aggregate has been replaced by the expression $f_{v\,N_{a_v}}(q,\ t,\ \textbf{PARTITION}(R,\ q,\ t,\ w_v,\ N_{a_v},\ X_v))$. With these changes, $\widehat{A}$ allows expressions to be aggregated. $\widehat{AU}$ can be modified similarly.

If $\widehat{A}$ and $\widehat{AU}$ are to support aggregation by arbitrary expressions, changes must be made to the definitions of **PARTITION**, $\widehat{A}$, and $\widehat{AU}$ given above. First, let $Evalue_l$, $1 \leq l \leq o$, be an expression involving the attribute names N_{c_1}, ..., N_{c_n}. $Evalue_l$ is evaluated for a tuple $r \in R$, by substituting the value components of the attributes of r for all occurrences of their corresponding attribute names in $Evalue_l$. Secondly, let $X = \{Evalue_1,\ \ldots,\ Evalue_o\}$ and $d_1,\ \ldots,\ d_p$ be the distinct integers in the range 1 to m such that N_{d_h}, $1 \leq h \leq p$, appears in at least one $Evalue_l, 1 \leq l \leq o$. Then new definitions of **PARTITION**, $\widehat{A}$, and $\widehat{AU}$ are constructed from the definitions above simply by substituting the predicate $\forall l,\ 1 \leq l \leq o\ (Evalue_l(r) = Evalue_l(q))$ for the predicate $\forall A \in X\ (value(r_A) = value(q_A))$ and the predicate $\forall l,\ 1 \leq l \leq p\ (valid(u_{d_l}) \neq \emptyset)$ for the predicate $\forall A \in X\ (valid(u_A) \neq \emptyset)$ in the definition of **PARTITION** and substituting p for n and $valid(q_{d_l})$ for $valid(q_A)$ in the definitions of $\widehat{A}$ and $\widehat{AU}$. With these changes, $\widehat{A}$ and $\widehat{AU}$ support aggregation by arbitrary expressions.

3.5 Closure, Completeness, and Reducibility

An important property of an algebra is that it is *closed*, that is, all operators produce valid objects, in this case historical relations.

Theorem 1 *The historical algebra is closed.*

PROOF. An historical relation is a finite set of historical tuples that satisfy three criteria. First, the values must be drawn from the appropriate domains, which is easy to show since values of an output attribute for all operators always originate from a single input attribute (or compatible attributes, in the case of binary operators). Second, at least one time-stamp in each tuple must be non-empty. This criterion is explicitly ensured in the semantics of $\hat{\sigma}$, $\hat{-}$, $\hat{\pi}$, $\hat{\delta}$ and AT via the **NotNull** function. Since $\hat{\times}$, $\widehat{A}$, and $\widehat{AU}$ append attributes to each input tuple, this criterion is satisfied. Because time-stamps remain the same or grow in $\hat{\cup}$, it also satisfies this criterion. Because

Here N_{agg} is the attribute name assigned the aggregate value $(y,\ \{t\})$. If X is not empty, function $\widehat{A}$ first associates with each time t the partition of relation Q whose tuples intersect the window in the valid component of attributes X and N (line 3). For each of these partitions, $\widehat{A}$ then constructs a set of historical tuples. Each tuple in the set contains all the attributes X of a tuple q in the partition and a new attribute. This new attribute's valid component is the time t corresponding to the partition and its value component is the scalar value returned by the aggregate f_N, when f_N is applied to the appropriate partition of R (line 4).

If X is empty, $\widehat{A}$ constructs for each time t an historical relation that is either empty or contains a single tuple. If the valid component of attribute N of no tuple r in R overlaps the window, then the historical relation is empty. Otherwise, the historical relation contains a single tuple whose valid component is the time t and whose value component is the scalar value returned by the aggregate f_N.

Note that a tuple and a time are passed as parameters to the scalar aggregate f_N, along with a partition of R, in the definition of $\widehat{A}$. Although most aggregate operators can be defined in terms of a single parameter, the partition of R, the additional parameters are present because aggregates that evaluate to events or intervals, one of which is defined in Section 4.3, require them.

3.4.3 Unique Aggregates

The function $\widehat{A}$ allows its embedded scalar aggregates to aggregate over duplicate attribute values. We now define an historical aggregate function $\widehat{AU}$, identical to $\widehat{A}$ with one exception; it restricts its embedded scalar aggregates to aggregation over *unique* attribute values.

EXAMPLE. Count uniquely the number of states in which enrolled students reside.

$$\widehat{AU}_{\text{COUNT},\,0,\,State,\,\emptyset}(\hat{\pi}_{State}(\text{S}_3),\ \text{S}_3) = \Big\{\ \langle(1,\ \{3,4,5,6,7,8\})\rangle,\ \langle(2,\ \{1,2\})\rangle\ \Big\}$$

This relation differs from the non-unique variant only during the interval $\{5,6\}$. Here, Virginia is correctly counted only once, even though there are two tuples valid during this interval with a state of Virginia. □

$$\begin{aligned}
&\widehat{AU}_{f,\,w,\,N_a,\,X}(Q,\ R) \triangleq \\
&\quad \hat{\pi}_{X\cup\{N_{agg}\}}\,(\{q \circ (y,\ z) \mid \quad q \in Q \\
&\qquad\qquad\qquad\qquad \wedge\ z = \bigcup\{t \mid \forall A \in (N \cup \{X\})\,((valid(q_A) \cap w\hat{}(t)) \neq \emptyset)\} \\
&\qquad\qquad\qquad\qquad \wedge\ \exists t \in z\,(y = f_N(q,\ t,\ \hat{\pi}_N(\textbf{PARTITION}(R,\ q,\ t,\ w,\ N,\ X))) \\
&\qquad\qquad \})
\end{aligned}$$

The formal definition differs from that of $\widehat{A}$ only in that the added historical projection on attribute N of **PARTITION**(...) eliminates duplicate values of the aggregated attribute before the scalar aggregation is preformed.

Only the third and fourth tuples in S_3 satisfy this requirement. The partitioning function here effectively returns the tuples for those students who were enrolled in school at time 5. Note that the time-stamp of each attribute in the selected tuples has been restricted to the interval from the attribute's original time-stamp overlapping time 5, if any.

Do the same, but group by State and select the group from Phil's state.

$$\textbf{PARTITION}(S_3,\ \langle(\text{Phil},\ \{1,3,4\}),\ (\text{Virginia},\ \{4,5,6\})\rangle,\ 5,\ 0,\ \mathit{Name},\ \{\mathit{State}\}) =$$
$$\{\ \langle(\text{Norman},\ \{5,6\}),\ (\text{Virginia},\ \{5,6\})\rangle\ \}$$

Here Q is assumed to be S_3; q is a tuple drawn from S_3. Tuples are selected for those students who were enrolled in school and a resident of Phil's state (Virginia) at time 5. Only the third tuple in S_3 satisfies this requirement. Although Phil was a resident of Virginia at time 5, he was not enrolled in school at time 5. Hence, the second tuple in S_3 is not included in this partition.

Do the same, except use a moving window of one year.

$$\textbf{PARTITION}(S_3,\ \langle(\text{Phil},\ \{1,3,4\}),\ (\text{Virginia},\ \{4,5,6\})\rangle,\ 5,\ 1,\ \mathit{Name},\ \{\mathit{State}\}) =$$
$$\{\ \langle(\text{Phil},\ \{3,4\}),\ (\text{Virginia},\ \{4,5,6\})\rangle$$
$$\langle(\text{Norman},\ \{5,6\}),\ (\text{Virginia},\ \{5,6\})\rangle\ \}$$

Here tuples are selected for those students who were enrolled in school and a resident of Virginia within a year ($w(t) = 1$) of time 5. Both the second and third tuples in S_3 satisfy this requirement. The second tuple in S_3 is now included in the partition because Phil was a resident of Virginia and enrolled in school at time 4. □

3.4.2 Non-unique Aggregates

The historical aggregate function $\widehat{A}_{f,\,w,\,N,\,X}(Q,\ R)$ calculates, for each tuple in Q, a distribution of scalar values over time for an arbitrary aggregate applied to attribute N of the subset of tuples in R whose value component for attributes X matches the value component for the same attributes of the tuple in Q. If X is empty, $\widehat{A}$ simply calculates a single distribution of scalar values over time for the aggregate applied to attribute N of R. f represents an arbitrary family of scalar aggregates and w represent an aggregation window function. The aggregate function is defined using the historical union and projection operators previously presented.

$$\widehat{A}_{f,\,w,\,N,\,X}(Q,\ R) \triangleq$$
$$\hat{\pi}_{X \cup \{N_{agg}\}}(\{q \circ (y,\ \{t\}) \mid \quad q \in Q$$
$$\wedge\ z = \bigcup\{t \mid \forall A \in (\{N\} \cup X)\,((\mathit{valid}(q_A) \cap w\hat{}(t)) \neq \emptyset)\}$$
$$\wedge\ \exists t \in z\,(y = f_N(q,\ t,\ \textbf{PARTITION}(R,\ q,\ t,\ w,\ N,\ X))$$
$$\})$$

EXAMPLE. For each state, count the number of enrolled students who reside in that state.

$$\widehat{A}_{\text{COUNT},\,0,\,Name,\,\{State\}}(\text{S}_3,\ \text{S}_3) = \Big\{\ \langle(\text{Kansas},\ \{1,2,3\}),\ (1,\ \{1,2,3\})\rangle$$
$$\langle(\text{Virginia},\ \{1,2,4,5,6\}),\ (1,\ \{1,2,4\})\rangle$$
$$\langle(\text{Virginia},\ \{1,2,4,5,6\}),\ (2,\ \{5,6\})\rangle$$
$$\langle(\text{Texas},\ \{7,8\}),\ (1,\ \{7,8\})\rangle\ \Big\}$$

Here, $\widehat{A}$ computes an instantaneous aggregate. In effect, the aggregate is computed for each subset of tuples in S_3 having the same value for the attribute *State*. For example, the first tuple is computed by selecting all the tuples in S_3 with a state of Kansas and then performing the aggregate on this (smaller) set. □

3.4.1 Partitioning Function

Before defining the historical aggregate functions $\widehat{A}$ and $\widehat{AU}$, we formally define a partitioning function. Recall that $X \subseteq \mathcal{N}$, $q \in Q$ is a tuple of the set of partitioning values, $t \in \mathcal{T}$, w is an aggregation window function, and $N \in \mathcal{N}_R$.

PARTITION$(R,\ q,\ t,\ w,\ N,\ X) \triangleq$

1 $\{u^m \mid \exists r \in R\,(\quad \forall A \in X\,(value(r_A) = value(q_A))$

2 $\qquad\qquad \wedge\ u \equiv r$

3 $\qquad\qquad \wedge\ \forall A \in \mathcal{N}\ (valid(u_A) = \{I \mid I \in \textbf{Interval}(valid(r_a)) \wedge (I \cap \hat{w}(t)) \neq \emptyset\})$

4 $\qquad\qquad \wedge\ valid(u_N) \neq \emptyset\ \wedge\ \forall A \in X\ (valid(u_A) \neq \emptyset)$

5 $\qquad\qquad)$

6 $\}$

This function retrieves from R those tuples that have the same value component for attributes in X as q (line 1) and have time t or some time in the interval of length $w(t)$ immediately preceding t in the time-stamp of attribute N (line 3).

EXAMPLES. Partition the entire S_3 *relation given on page 6 at the instant 5 based on the Name attribute.*

$$\textbf{PARTITION}(\text{S}_3,\ \langle\ \rangle,\ 5,\ 0,\ Name,\ \emptyset) = \Big\{\ \langle(\text{Norman},\ \{5,6\}),\ (\text{Virginia},\ \{5,6\})\rangle$$
$$\langle(\text{Norman},\ \{5,6\}),\ (\text{Texas},\ \emptyset)\rangle\ \Big\}$$

Because time 5 is specified and the aggregation window function, denoted by zero, is the constant function $w(t) = 0$, tuples are selected whose time-stamp for attribute $Name$ overlaps time 5.

languages (e.g., the attributes in the `GROUP BY` clause in SQL [IBM 1981]). Generally $X = \mathcal{N}_Q$ and $Q = \pi_X(R)$, but these constraints are not dictated by the formal definition of $\widehat{A}$.

Finally, assume, as does Klug, that for each aggregate operation (e.g., `count`) we have a family of scalar aggregates that performs the indicated aggregation on R (e.g., COUNT_{N_1}, COUNT_{N_2}, ..., COUNT_{N_m}, where COUNT_{N_a} counts the (possibly duplicate) values of attribute N_a of R). The particular scalar aggregate is denoted by f. w represents an aggregation window function.

EXAMPLE. Count the number of states in which enrolled students resided.

$$\widehat{A}_{\text{COUNT},\,0,\,State,\,\emptyset}(\hat{\pi}_{State}(\text{S}_3),\ \text{S}_3) = \{\ \langle(1,\ \{3,4,7,8\})\rangle,\ \langle(2,\ \{1,2,5,6\})\rangle\ \}$$

Because $w(t) = 0$ and the time granularity of S_3 is a semester, the resulting relation represents aggregation by semester. Hence, the aggregate is in effect an instantaneous aggregate. For the interval $\{1,2\}$, there were two states (Kansas in the first tuple and Virginia in the third tuple). For the interval $\{3,4\}$, there was one state (Kansas in the first tuple at time 3 and Virginia in the second tuple at time 4). For the interval $\{5,6\}$, there also was only one state (Virginia), but it appeared in both the second and third tuples. It was counted twice because the scalar aggregates embedded within $\widehat{A}$ aggregate over duplicate values. For the interval $\{7,8\}$, there was only one state (Texas, in the fourth tuple). □

EXAMPLE. Count the number of states in which enrolled students reside over the previous year.

$$\widehat{A}_{\text{COUNT},\,1,\,State,\,\emptyset}(\hat{\pi}_{State}(\text{S}_3),\ \text{S}_3) = \{\ \langle(1,\ \{8,9\})\rangle,\ \langle(2,\ \{1,2,3,4,5,6\})\rangle,\ \langle(3,\ \{7\})\rangle\ \}$$

Here, $w(t) = 2 - 1 = 1$ representing aggregation by year (assuming two semesters per year). Although the chronon 9 does not appear in the time-stamp of attribute *State* in any tuple in S_3, a count of one is recorded at time 9 because a tuple, the fourth tuple in S_3, falls into the aggregation window at time 9. □

EXAMPLE. Count the number of states in which enrolled students ever resided.

$$\widehat{A}_{\text{COUNT},\,\infty,\,State,\,\emptyset}(\hat{\pi}_{State}(\text{S}_3),\ \text{S}_3) = \{\ \langle(2,\ \{1,2,3\})\rangle,\ \langle(3,\ \{4,5,6\})\rangle,\ \langle(4,\ \{7,8,\ldots\})\rangle\ \}$$

With $w(t) = \infty$, $\widehat{A}$ computes a cumulative aggregate. □

terms of the time granularity. Hence, instantaneous aggregates can be viewed as aggregates over an interval whose duration is determined by the granularity of the measure of time being used. Others have generalized the definition of instantaneous and cumulative aggregates by introducing the concept of *moving aggregation windows* [Navathe & Ahmed 1989]. An *aggregation window function* w is a function from the domain $\mathcal{T}$ into the non-negative integers. For each time t, the window function w gives rise to the interval of length $w(t)$ immediately preceding time t. This interval, termed the *window* ($\hat{w}$), may be defined as

$$\hat{w} \triangleq \textbf{Extend}(\textbf{Last}(1,\ t - w(t)),\ t)$$

An aggregate returns, for each time t, a value computed from tuples valid during the window. An instantaneous aggregate is an aggregate with an aggregation window function $w(t) = 0$ and a cumulative aggregate is an aggregate with an aggregation window function $w(t) = \infty$.

Klug introduced an approach to handle aggregates in the snapshot algebra [Klug 1982]. His approach makes it possible to define aggregates, in particular, non-unique aggregates, in a rigorous fashion. We use his approach to define two historical aggregate functions for the algebra.

- $\widehat{A}$, that calculates non-unique aggregates, and
- $\widehat{AU}$, that calculates unique aggregates.

These two historical aggregate functions serve as the historical counterpart of both scalar aggregates and aggregate functions.

The historical aggregate functions must contend with a variety of demands that surface as parameters (subscripts) to the functions. First, a specific aggregate (e.g., `count`) must be specified. Secondly, the attribute over which the aggregate is to be applied must be stated and the aggregation window function must be indicated. Finally, to accommodate partitioning, where the aggregate is applied to partitions of a relation, a set of partitioning attributes must be given. These demands complicate the definitions of $\widehat{A}$ and $\widehat{AU}$, but at the same time ensure some degree of generality to these operators.

The aggregate operator is denoted by $\widehat{A}_{f,\ w,\ N,\ X}(Q,\ R)$. R is an historical relation of m-tuples over the relation scheme $\mathcal{N}_R$. $N \in \mathcal{N}_R$ is the attribute on which the aggregate is applied. Q supplies the values that partition R. X denotes the attributes on which the partitioning is applied, with the restrictions that $\mathcal{N}_Q \subseteq \mathcal{N}_R$ and $\{N\} \cup X \subseteq \mathcal{N}_Q$.

If X is empty, the historical aggregate operators simply calculate a single distribution of scalar values over time for an arbitrary aggregate applied to attribute N of relation R. In this case, the tuples in Q are ignored.

If X is not empty, the operators calculate, for each subtuple in Q formed from the attributes X, a distribution of scalar values over time for an aggregate applied to attribute N of the subset of tuples in R whose values for attributes X match the values for the same attributes of the tuple in Q. Hence, X corresponds to the by-list of an aggregate function in conventional database query

derivation operator could have been replaced by two simpler operators, analogous to the selection and projection operators, that would have performed tuple selection and attribute projection in terms of the valid components, rather than the value components, of attributes. But, as we will see in Section 6, disallowing set-valued time-stamps would have required that the algebra support value-equivalent tuples, which would have prevented the algebra from having several other, more highly desirable properties.

3.3 Conversion Operators

The snapshot operator $\widehat{SN}$ computes a snapshot relation valid at a specified time τ. If only a subset of attributes is valid at τ, that tuple is not selected.

$$\widehat{SN}(R,\ \tau) = \{(value(r_1),\ \ldots,\ value(r_n)) \mid \exists r \in R\ \forall A \in \mathcal{N}(\tau \in valid(r_A))\}$$

EXAMPLE. $S_4 = \widehat{SN}(S_3,\ 4) = \{(\text{Phil},\ \text{Virginia})\}$ □

The dual is the AT operator, which converts a snapshot relation to its historical analogue at considered valid at the specified time τ.

$$AT(R',\ \tau) = \{r \mid \exists r' \in R'\, \forall A \in \mathcal{N}\ (r'_A = value(r_A) \wedge\ valid(r_A) = \tau)\}$$

EXAMPLE. $S_4 = AT(S_4,\ 4) = \{\ \langle(\text{Phil}, \{4\}),\ (\text{Virginia},\ \{4\})\rangle$ □

3.4 Aggregates

Aggregates allow users to summarize information contained in a relation. Aggregates are categorized as either scalar aggregates or aggregate functions. *Scalar aggregates* return a single scalar value that is the result of applying the aggregate to a specified attribute of a snapshot relation. *Aggregate functions*, however, return a set of scalar values, each value the result of applying the aggregate to a specified attribute of those tuples in a snapshot relation having the same values for certain attributes. Database management systems based on the relational model typically provide several aggregate operators. For example, Quel [Stonebraker et al. 1976] provides a `count`, `sum`, `average`, `minimum`, `maximum`, and `any` aggregate operator. Quel also provides two versions of the `count`, `sum`, and `average` operators, one that aggregates over all values of an attribute and one that aggregates over only the unique values of an attribute.

Several researchers have investigated aggregates in time-oriented relational databases [Ben-Zvi 1982, Jones et al. 1979, Navathe & Ahmed 1989, Snodgrass et al. 1989, Tansel et al. 1989]. Their work reflects the consensus that aggregates when applied to historical relations should return not a single value, but a distribution of values over time. Jones, et al. also introduced the concepts of *instantaneous aggregates* and *cumulative aggregates*. Instantaneous aggregates return, for each time t, a value computed only from the tuples valid at time t. Cumulative aggregates return, for each time t, a value computed from all tuples valid at any time up to and including t, regardless of whether the tuples are still valid at time t. Note that a time t has meaning only when defined in

in the linear ordering of all times.

Succ is the successor function on the domain $\mathcal{T}$. It maps a time into its immediate successor in the linear ordering of all times.

Extend maps two times into the set of times that represents the interval between the first time and the second time.

Interval maps a set of times into the set of intervals containing the minimum number of non-disjoint intervals represented by the input set. Each time in the input set appears in exactly one interval in the output set and each interval in the output set is itself represented by a set of times.

EXAMPLE. Consider the following tuple taken from the relation S_3 defined previously.

$$r = \langle(\text{Norman}, \{1,2,5,6\}), (\text{Texas}, \{7,8\})\rangle$$

then

$$\begin{aligned} \textbf{Interval}(\mathit{valid}(r(\mathit{Name}))) &= \big\{\{1,2\}, \{5,6\}\big\} \\ \textbf{Interval}(\mathit{valid}(r(\mathit{State}))) &= \big\{\{7,8\}\big\} \end{aligned}$$

□

Let R be an historical relation of m-tuples. Let V_a, $1 \leq a \leq m$, be temporal functions involving the attribute names $N_1, \ldots, N_m$, constants from the domain $\mathcal{I}$ of non-disjoint intervals defined in Appendix B, the functions **First**, **Last**, and **Extend**, and the set operators $\cup$, $\cap$, and $-$. Let G be a boolean function involving the temporal functions, as just described, the relational operators $<$, $=$, and $>$, and the logical operators $\wedge$, $\vee$, and $\neg$.

Apply selects an interval from the temporal element of each attribute's timestamp, applies the predicate G to these intervals, and, if G returns true, evaluates the V_i to generate an output interval.

$$\begin{aligned} \textbf{Apply}(G, V_1, \ldots, V_m, R) = \{u \mid \exists r \in R\,(u \equiv r \wedge\ & \exists I_i \in \textbf{Interval}(\mathit{valid}(r_i)),\ 1 \leq i \leq m \\ & (G(I_1, \ldots, I_m) \wedge\ \mathit{valid}(u_i) = V_i(I_1, \ldots, I_m))\} \end{aligned}$$

Note that the resulting set may contain many value-equivalent tuples.

Given these auxiliary functions, we can now define the historical derivation operator on historical relations. In the formal semantics, the functions G and V_a are always evaluated for a specific assignment of non-disjoint intervals to attribute names $N_1, \ldots, N_m$. G evaluates to either true or false and V_a evaluates to an interval.

$$\hat{\delta}_{G, V_1, \ldots, V_m}(R) \triangleq \textbf{NotNull}(\textbf{Reduce}(\textbf{Apply}(G, V_1, \ldots, V_m, R)))$$

The **Apply** function, and thus the derivation operator, is necessarily somewhat complex because we allow set-valued time-stamps. Had we had disallowed set-valued time-stamps, the

time-stamp for this tuple's attributes would be assigned the empty set in the resulting relation except the definition of the historical derivation operator disallows tuples whose attributes all have an empty time-stamp. This tuple is therefore eliminated and does not appear in the resulting relation. From the third tuple, $\langle$(Norman, $\{1,2,5,6\}$), (Virginia, $\{1,2,5,6\}$)$\rangle$, we find that Norman was a resident of Virginia during both of his intervals of enrollment ($G(\{1,2\},\ \{1,2\})$ is true and $G(\{5,6\},\ \{5,6\})$ is true). Hence, this tuple's attributes are assigned a time-stamp of $\{1,2,5,6\}$ in the resulting relation. From the fourth tuple, $\langle$(Norman, $\{1,2,5,6\}$), (Texas, $\{7,8\}$)$\rangle$, we find that Norman was not a resident of Texas at any time during his enrollment ($G(\{1,2\},\ \{7,8\})$ is false and $G(\{5,6\},\ \{7,8\})$ is false); this tuple is therefore eliminated from the resulting relation. □

EXAMPLE. Find the students who home state changed during at least one interval of enrollment.

$$\hat{\delta}_{(Name \cap State) \neq Name \ \wedge \ (Name \cap State) \neq \emptyset,\ Name \cap State}(\mathrm{S}_3) = \Big\{ \begin{array}{l} \langle(\text{Phil, } \{3\}), (\text{Kansas, } \{3\})\rangle, \\ \langle(\text{Phil, } \{4\}), (\text{Virginia, } \{4\})\rangle \end{array} \Big\}$$

The new time-stamp for each tuple that satisfies G for some assignment of intervals $Name$ and $State$ contains the intervals $Name \cap State$ from each assignment of intervals that satisfy G. From the first tuple in S_3 we find that Phil had one interval of enrollment during which his home state changed, because $G(\{3,\ 4\},\ \{1,\ 2,\ 3\}) = (\{3,4\} \cap \{1,2,3\} \neq \{3,4\} \wedge \{3,4\} \cap \{1,2,3\} \neq \emptyset)$ evaluates to true. Hence, this tuple's attributes are assigned a time-stamp of $\{3,4\} \cap \{1,2,3\} = \{3\}$ in the resulting relation. From the second tuple in S_3 we find that Phil had one interval of enrollment during which his home state changed. Hence, this tuple's attributes are assigned a time-stamp of $\{4\}$ in the resulting relation. Note that Norman does not satisfy the restriction; his home state was the same during his two periods of enrollment. Hence, the third and fourth tuples are eliminated from the resulting relation. □

Note that the historical derivation operator actually performs two functions. First, it performs a selection function on the valid component of a tuple's attributes. For a tuple r, if G is false when an interval from the valid component of each of r's attributes is substituted for each occurrence of its corresponding attribute name in G, then the temporal information represented by that combination of intervals is not used in the calculation of the new time-stamps for r's attributes. Secondly, the derivation operator calculates a new time-stamp for attribute N_a, $1 \leq a \leq m$, from those combinations of intervals for which G is true, using V_a. If V_1, ..., V_m are all the same function, the tuple is effectively converted from attribute time-stamping to tuple time-stamping.

Several functions, defined on the domains $\mathcal{T}$ and $\wp(\mathcal{T})$, are used either directly or indirectly in the definition of the historical derivation operator. Before defining the derivation operator itself, we describe informally these auxiliary functions. Formal definitions appear in Appendix B.

First takes a set of times from the domain $\wp(\mathcal{T})$ and maps it into the earliest time in the set.

Last takes a set of times from the domain $\wp(\mathcal{T})$ and maps it into the latest time in the set.

Pred is the predecessor function on the domain $\mathcal{T}$. It maps a time into its immediate predecessor

signment of intervals to attribute names for which G is true, the operator evaluates V_a, $1 \leq a \leq m$. The sets of times resulting from the evaluations of V_a are then combined to form a new time-stamp for attribute N_a. For notational convenience, we assume that if only one V-function is provided, it applies to all attributes.

EXAMPLE. List the state of residence for the periods when each student was enrolled.

$$\begin{aligned} \mathit{HomeWhenEnrolled} = \hat{\delta}_{G,V}(\mathrm{S}_3) = \{ \; & \langle(\text{Phil}, \{1,3\}), (\text{Kansas}, \{1,3\})\rangle, \\ & \langle(\text{Phil}, \{4\}), (\text{Virginia}, \{4\})\rangle, \\ & \langle(\text{Norman}, \{1,2,5,6\}), (\text{Virginia}, \{1,2,5,6\})\rangle \; \} \end{aligned}$$

where

$$G \equiv \mathbf{Last}(\mathbf{First}(\mathit{SName}), \mathbf{First}(\mathit{State})) < \mathbf{First}(\mathbf{Last}(\mathit{SName}), \mathbf{Last}(\mathit{State}))$$

$$V \equiv \mathbf{Extend}(\mathbf{Last}(\mathbf{First}(\mathit{SName}), \mathbf{First}(\mathit{State})), \; \mathbf{First}(\mathbf{Last}(\mathit{SName}), \mathbf{Last}(\mathit{State})))$$

Here **First** selects the earliest chronon from an interval; **Last** selects the latest chronon; and **Extend**(a, b) generates an interval commencing with a and ending at b. In this example, G determines whether the interval of residency overlaps the interval of enrollment (it may take the reader a moment to be convinced that G is indeed merely overlap), and V calculates this overlap, i.e., the interval during which both were valid. This interval is assigned to both attributes. □

EXAMPLE. Find the periods of enrollment during which the student's home state did not change.

$$\begin{aligned} \hat{\delta}_{(\mathit{Name} \cap \mathit{State}) = \mathit{Name}, \, \mathit{Name}, \, \mathit{Name}}(\mathrm{S}_3) = \{ \; & \langle(\text{Phil}, \{1\}), (\text{Kansas}, \{1\})\rangle, \\ & \langle(\text{Norman}, \{1,2,5,6\}), (\text{Virginia}, \{1,2,5,6\})\rangle \; \} \end{aligned}$$

In this example, G is "$(\mathit{Name} \cap \mathit{State}) = \mathit{Name}$" and V_1 and V_2 are both "Name". A student tuple $s \in \mathrm{S}_3$ satisfies condition G if the student had at least one interval of enrollment during which his home state (i.e, *State*) did not change. The new time-stamp for each attribute of a tuple that satisfies G for some assignment of intervals Name and State consists of the Name intervals from each assignment of intervals that satisfy G. In the first tuple in S_3, $\langle(\text{Phil}, \{1,3,4\}), (\text{Kansas}, \{1,2,3\})\rangle$, there are three intervals, two assigned to the attribute *Name* ($\{1\}$, $\{3,4\}$) and one assigned to the attribute *State* ($\{1,2,3\}$). From this tuple, we find that Phil was a resident of Kansas during his first interval of enrollment, because $G(\{1\}, \{1,2,3\}) = (\{1\} \cap \{1,2,3\} = \{1\})$ evaluates to true. However, Phil was a resident of Kansas during only part of his second interval of enrollment, because $G(\{3,4\}, \{1,2,3\})$ is false. Hence, this tuple's attributes are assigned a time-stamp of $\{1\}$ in the resulting relation. From the second tuple in S_3, $\langle(\text{Phil}, \{1,3,4\}), (\text{Virginia}, \{4,5,6\})\rangle$, we find that Phil was not a resident of Virginia during his first interval of enrollment ($G(\{1\}, \{4,5,6\})$ is false) and lived in Virginia during only part of his second interval of enrollment ($G(\{3,4\}, \{4,5,6\})$ is false). Hence, the

The $\hat{\pi}$ operator also supports projections on expressions. For an arbitrary n, let $Evalue_l$, $1 \leq l \leq n$, be an arbitrary expression involving the attribute names N_a, $1 \leq a \leq m$. $Evalue_l$ is evaluated, for a tuple $r \in R$, by substituting the value components of the attributes of r for all occurrences of their corresponding attribute names in $Evalue_l$. Also, let $Evalid_l$, $1 \leq l \leq n$, be an arbitrary expression involving the attribute names N_a, $1 \leq a \leq m$, where $Evalid_l$ is evaluated for a tuple $r \in R$ by substituting the valid components of the attributes of r for all occurrences of their corresponding attribute names in $Evalid_l$. In addition, assume that evaluation of $Evalue_l$ for every tuple r produces an element of the domain $\mathcal{D}_b$, $1 \leq b \leq m$, and that evaluation of $Evalid_l$ produces an element of the domain $\wp(\mathcal{T})$. Then the definition of $\hat{\pi}$, now denoted by $\hat{\pi}_{(Evalue_1,\, Evalid_1),\, ...,\, (Evalue_n,\, Evalid_n)}(R)$, is constructed from the definition above simply by substituting $Evalue_h(r)$ for $value(r_{a_h})$, $Evalid_h(r)$ for $valid(r_{a_h})$, $Evalue_l(r)$ for $value(r_{a_l})$, and $Evalid_l(r)$ for $valid(r_{a_l})$. Note that this definition of the $\hat{\pi}$ operator is simply a more general version of the definition presented earlier, where N_{a_l}, $1 \leq l \leq n$, is assumed to be the ordered pair of expressions $(N_{a_l},\, N_{a_l})$.

EXAMPLE.

$$\hat{\pi}_{(name,\,(sname,\,sname)),\,(state,\,(state,\,sname \cap state))}(\mathrm{S}_3) =$$

$$\begin{Bmatrix} \langle(\text{Phil},\ \{1,3,4\}),\ (\text{Kansas},\ \{1,3\})\rangle, \\ \langle(\text{Phil},\ \{1,3,4\}),\ (\text{Virginia},\ \{4\})\rangle, \\ \langle(\text{Norman},\ \{1,2,5,6\}),\ (\text{Virginia},\ \{1,2,5,6\})\rangle, \\ \langle(\text{Norman},\ \{1,2,5,6\}),\ (\text{Texas},\ \emptyset)\rangle \end{Bmatrix}$$

The result is a historical relation with attributes {`name`, `state`} rather than {`sname`, `state`}. The valid-time component of attribute `name` represents the interval(s) when the specified student was enrolled, but the valid-time component of attribute `state` represents only the subinterval(s) of enrollment when the student was a resident of the specified state. Note that, because Norman's enrollment never overlapped his residency in Texas, the valid-time component of the attribute `state` of the fourth tuple is the empty set. □

3.2 Historical Derivation

The historical derivation operator $\hat{\delta}$ is a new operator that does not have an analogous snapshot operator. It replaces the time-stamp of each attribute in a tuple with a new time-stamp, where the new time-stamps are computed from the existing time-stamps of the tuple's attributes. $\hat{\delta}$ is effectively a combination of selection and projection on a tuple's attribute time-stamps.

$\hat{\delta}_{G,\, V_1,\, ...,\, V_m}(R)$ determines, for a tuple $r \in R$, new time-stamps for r's attributes. The historical derivation function first determines all possible assignments of *intervals* to attribute names for which the predicate G on time-stamps is true. Hence, an occurrence of an attribute A in G and in V is intended to be a variable, which evaluates to an interval upon tuple substitution. For each as-

$\langle$(Norman, {5,6}), (Calculus, {5,6}), (Phil, {4,5,6}), (Virginia, {4,5,6})$\rangle$,
$\langle$(Norman, {5,6}), (Calculus, {5,6}), (Norman, {1,2,5,6}), (Virginia, {1,2,5,6})$\rangle$,
$\langle$(Norman, {5,6}), (Calculus, {5,6}), (Norman, {7,8}), (Texas, {7,8})$\rangle$ $\}$ □

Let R be an historical relation of m-tuples. Also, let F be a boolean function involving the attribute names $N_1, \ldots, N_m$, constants from the domains $Dom(N_1), \ldots, Dom(N_m)$, the relational operators $<$, $=$, and $>$, and the logical operators $\wedge$, $\vee$, and $\neg$. To evaluate F for a tuple $r \in R$, we substitute the value components of the attributes of r for all occurrences of their corresponding attribute names in F. Then the historical selection $\hat{\sigma}_F(R)$ is identical to selection in the snapshot algebra: it evaluates to the set of tuples in R for which F is true.

$$\hat{\sigma}_F(R) \triangleq \{r^m \mid r \in R \wedge F(value(r_1), \ldots, value(r_m))\}$$

EXAMPLE. $S_2 = \hat{\sigma}_{SName=HName}(S_1) =$

$\{$ $\langle$(Phil, {1,3,4}), (English, {1,3,4}), (Phil, {1,2,3}), (Kansas, {1,2,3})$\rangle$,
$\langle$(Phil, {1,3,4}), (English, {1,3,4}), (Phil, {4,5,6}), (Virginia, {4,5,6})$\rangle$,
$\langle$(Norman, {1,2}), (English, {1,2}), (Norman, {1,2,5,6}), (Virginia, {1,2,5,6})$\rangle$,
$\langle$(Norman, {1,2}), (English, {1,2}), (Norman, {7,8}), (Texas, {7,8})$\rangle$,
$\langle$(Norman, {5,6}), (Calculus, {5,6}), (Norman, {1,2,5,6}), (Virginia, {1,2,5,6})$\rangle$,
$\langle$(Norman, {5,6}), (Calculus, {5,6}), (Norman, {7,8}), (Texas, {7,8})$\rangle$ $\}$

□

Let R be an historical relation of m-tuples and let $a_1, \ldots, a_n$ be distinct integers in the range 1 to m. Like the projection operator for snapshot relation, the projection operator for historical relations, $\hat{\pi}_{N_{a_1}, \ldots, N_{a_n}}$, retains, for each tuple, the tuple components that correspond to the attribute names $N_{a_1}, \ldots, N_{a_n}$.

$$\hat{\pi}_{N_{a_1}, \ldots, N_{a_n}}(R) \triangleq \textbf{Reduce}(\textbf{NotNull}(\{u^n \mid \exists r \in R \, \forall i, 1 \leq i \leq n \, (u_i = r_{a_i})\}))$$

EXAMPLE. $S_3 = \hat{\pi}_{SName, State}(S_2) =$ $\{$ $\langle$(Phil, {1,3,4}), (Kansas, {1,2,3})$\rangle$,
$\langle$(Phil, {1,3,4}), (Virginia, {4,5,6})$\rangle$,
$\langle$(Norman, {1,2,5,6}), (Virginia, {1,2,5,6})$\rangle$,
$\langle$(Norman, {1,2,5,6}), (Texas, {7,8})$\rangle$ $\}$

We interpret S_3 as follows. The time-stamp associated with the value of the attribute *Name* represents the interval(s) when the specified student was enrolled and the time-stamp associated with the value of the attribute *State* represents the interval(s) when the student was a resident of the specified state. □

$$
\begin{aligned}
&\mathit{Courses}\,\hat{\cup}\ \{\ \langle(\text{Phil},\ \{4,5\}),\ (\text{English},\ \{4,5\})\rangle\ \} \\
&\quad = \{\ \langle(\text{Phil},\ \{1,3,4,5\}),\ (\text{English},\ \{1,3,4,5\})\rangle, \\
&\qquad \langle(\text{Norman},\ \{1,2\}),\ (\text{English},\ \{1,2\})\rangle, \\
&\qquad \langle(\text{Norman},\ \{5,6\}),\ (\text{Calculus},\ \{5,6\})\rangle\ \}\quad \square
\end{aligned}
$$

The historical difference of Q and R, $Q \hat{-} R$, is the set of all tuples such that the time-stamp of each attribute of a tuple in $Q \hat{-} R$ must equal the set difference of the time-stamps of the corresponding attribute in the value-equivalent tuple in Q and the value-equivalent tuple in R.

$$
\begin{aligned}
Q \hat{-} R \triangleq\ & \{q^m \mid \exists q \in Q \wedge \neg(\exists r \in R\ (r \equiv q))\} \\
& \bigcup \mathbf{NotNull}(\{u^m \mid\ \exists q \in Q\ \exists r \in R\ (u \equiv q \equiv r \wedge \forall A \in \mathcal{N}\ (\mathit{valid}(u_A) = \mathit{valid}(q_A) - \mathit{valid}(r_A)))\})
\end{aligned}
$$

EXAMPLE. Phil didn't take English during semesters 4 and 5.

$$
\begin{aligned}
&\mathit{Courses}\,\hat{-}\ \{\ \langle(\text{Phil},\ \{4,5\}),\ (\text{English},\ \{4,5\})\rangle\ \} \\
&\quad = \{\ \langle(\text{Phil},\ \{1\}),\ (\text{English},\ \{1\})\rangle, \\
&\qquad \langle(\text{Norman},\ \{1,2\}),\ (\text{English},\ \{1,2\})\rangle, \\
&\qquad \langle(\text{Norman},\ \{5,6\}),\ (\text{Calculus},\ \{5,6\})\rangle\ \}\quad \square
\end{aligned}
$$

Now let Q be an historical relation of m_1-tuples and R be an historical relation of m_2-tuples. Because historical relations are attribute-value time-stamped, cartesian product is a particularly simple operator. The historical cartesian product is identical to that for snapshot relations. In the following, "$\circ$" denotes concatenation.

$$Q \hat{\times} R \triangleq\ \{q \circ r \mid \exists q \in Q\ \wedge\ \exists r \in R\}$$

EXAMPLE. Home is an historical relation over the relation scheme $\mathit{Home} = \{\mathit{Name}, \mathit{State}\}$.

$$
\begin{aligned}
\mathit{Home} = \ \{\ & \langle(\text{Phil},\ \{1,2,3\}),\ (\text{Kansas},\ \{1,2,3\})\rangle, \\
& \langle(\text{Phil},\ \{4,5,6\}),\ (\text{Virginia},\ \{4,5,6\})\rangle, \\
& \langle(\text{Norman},\ \{1,2,5,6\}),\ (\text{Virginia},\ \{1,2,5,6\})\rangle, \\
& \langle(\text{Norman},\ \{7,8\}),\ (\text{Texas},\ \{7,8\})\rangle\ \}
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{S}_1 = \mathit{Courses}\,\hat{\times}\,\mathit{Home} = \\
&\quad \{\ \langle(\text{Phil},\ \{1,3,4\}),\ (\text{English},\ \{1,3,4\}),\ (\text{Phil},\ \{1,2,3\}),\ (\text{Kansas},\ \{1,2,3\})\rangle, \\
&\qquad \langle(\text{Phil},\ \{1,3,4\}),\ (\text{English},\ \{1,3,4\}),\ (\text{Phil},\ \{4,5,6\}),\ (\text{Virginia},\ \{4,5,6\})\rangle, \\
&\qquad \langle(\text{Phil},\ \{1,3,4\}),\ (\text{English},\ \{1,3,4\}),\ (\text{Norman},\ \{1,2,5,6\}),\ (\text{Virginia},\ \{1,2,5,6\})\rangle, \\
&\qquad \langle(\text{Phil},\ \{1,3,4\}),\ (\text{English},\ \{1,3,4\}),\ (\text{Norman},\ \{7,8\}),\ (\text{Texas},\ \{7,8\})\rangle, \\
&\qquad \langle(\text{Norman},\ \{1,2\}),\ (\text{English},\ \{1,2\}),\ (\text{Phil},\ \{1,2,3\}),\ (\text{Kansas},\ \{1,2,3\})\rangle, \\
&\qquad \langle(\text{Norman},\ \{1,2\}),\ (\text{English},\ \{1,2\}),\ (\text{Phil},\ \{4,5,6\}),\ (\text{Virginia},\ \{4,5,6\})\rangle, \\
&\qquad \langle(\text{Norman},\ \{1,2\}),\ (\text{English},\ \{1,2\}),\ (\text{Norman},\ \{1,2,5,6\}),\ (\text{Virginia},\ \{1,2,5,6\})\rangle, \\
&\qquad \langle(\text{Norman},\ \{1,2\}),\ (\text{English},\ \{1,2\}),\ (\text{Norman},\ \{7,8\}),\ (\text{Texas},\ \{7,8\})\rangle, \\
&\qquad \langle(\text{Norman},\ \{5,6\}),\ (\text{Calculus},\ \{5,6\}),\ (\text{Phil},\ \{1,2,3\}),\ (\text{Kansas},\ \{1,2,3\})\rangle,
\end{aligned}
$$

3 Historical Operators

We present ten operators that serve to define the historical algebra. Five of these operators — union, difference, cartesian product, projection, and selection — are analogous to the five operators that serve to define the snapshot algebra for snapshot relations [Ullman 1988]. Each of these five operators on historical relations is represented as $\hat{op}$ to distinguish it from its snapshot algebra counterpart op. Historical derivation is a new operator that replaces the time-stamp of each attribute value in a tuple with a new time-stamp, where the new time-stamps are computed from the existing time-stamps of the tuple's attributes. The snapshot ($\widehat{SN}$) and AT operators convert between historical and snapshot relations. The two remaining operators are aggregation and unique aggregation. After defining the operators, we show that the historical algebra is closed, is complete, and reduces to the snapshot algebra.

We will use two auxiliary functions in the formal semantics. Both operate over a set of tuples R.

$$\mathbf{NotNull}(R) \triangleq \{R \mid \exists r \in R \wedge \exists A \in \mathcal{N}\,(valid(R_A) \neq \emptyset)\}$$

This function ensures that all tuples in the resulting relation have at least one attribute value which has a non-null timestamp.

$$\begin{aligned}\mathbf{Reduce}(R) \triangleq \{U^n \mid \quad & \forall A \in \mathcal{N}\, \exists r \in R(r \equiv u \wedge \forall t \in valid(u_A)(t \in valid(r_A))) \\ & \wedge \forall r \in R(r \equiv u \Rightarrow \forall A \in \mathcal{N}(valid(r_A) \subseteq valid(u_A)))\}\end{aligned}$$

Reduce computes the minimal set of value-equivalent tuples, i.e., the set for which there are no such tuples. The first line ensures that no chronons have been manufactured; the second line ensures that all chronons of R are accounted for.

Thus, if R is a set of tuples over $\mathcal{N}$, then $\mathbf{Reduce}(\mathbf{NotNull}(R))$ is an historical relation.

3.1 Historical Analogues of the Snapshot Operators

Let Q and R be historical relations of m-tuples over the same relation scheme. Then the historical union of Q and R, $Q \hat{\cup} R$, is the set of tuples that are only in Q, are only in R, or are in both relations, with the restriction that each pair of value-equivalent tuples is represented by a single tuple. The time-stamp associated with each attribute of this tuple in $Q \hat{\cup} R$ is the set union of the time-stamps of the corresponding attribute in the value-equivalent tuples in Q and R.

$$Q \hat{\cup} R \triangleq \mathbf{Reduce}(\{u \mid u \in Q \vee u \in R\})$$

EXAMPLE. Add the fact that Phil took English in semesters 4 and 5.

Hereafter, we will refer to $ht(A)$ simply as ht_A, where A denotes an attribute in scheme $\mathcal{N}$, when there is no ambiguity of meaning. Note that it is possible for all but one attribute to have an empty time-stamp.

Two tuples, ht and ht', are said to be *value-equivalent* ($ht \equiv ht'$) if and only if $\forall A \in \mathcal{N}$, $value(ht_A) = value(ht'_A)$. An *historical relation* h is then defined as a finite set of historical tuples, with the restriction that no two tuples in the relation are value-equivalent.

EXAMPLE. Assume that we are given the relation scheme $Student = \{Name, Course\}$ and the following set of tuples over this relation scheme. For this and all later examples, assume that the granularity of time is a semester relative to the Fall semester 1980. Hence, 1 represents the Fall semester 1980, 2 represents the Spring semester 1981, etc. This relation specifies the courses each student took.

$$\begin{aligned} \mathrm{S} = \{ \; & \langle(\text{Phil}, \{1,3\}), (\text{English}, \{1,3\})\rangle, \\ & \langle(\text{Norman}, \{1,2\}), (\text{English}, \{1,2\})\rangle, \\ & \langle(\text{Norman}, \{5,6\}), (\text{Calculus}, \{5,6\})\rangle, \\ & \langle(\text{Phil}, \{4\}), (\text{English}, \{4\})\rangle \; \} \end{aligned}$$

For notational convenience we enclose each attribute value in parentheses and each tuple in angular brackets (i.e., $\langle\ \rangle$). We assume the natural mapping between attribute names and attribute values (e.g., $Name \rightarrow$ (Phil, $\{1,3\}$), and $Course \rightarrow$ (English, $\{1,3\}$)). Note that S is not an historical relation because there are value-equivalent tuples in the set (the first and fourth tuples are value-equivalent). If we replace the two value-equivalent tuples in S with a single tuple, then the new set *Courses* is an historical relation. While the attributes of a tuple in *Courses* have the same time-stamp, in general attributes within a tuple can have different time-stamps.

$$\begin{aligned} Courses = \{ \; & \langle(\text{Phil}, \{1,3,4\}), (\text{English}, \{1,3,4\})\rangle, \\ & \langle(\text{Norman}, \{1,2\}), (\text{English}, \{1,2\})\rangle, \\ & \langle(\text{Norman}, \{5,6\}), (\text{Calculus}, \{5,6\})\rangle \; \} \end{aligned}$$ □

In summary, the historical algebra places the same basic restrictions on the value components of attributes as the snapshot algebra places on attribute values. Neither set-valued attribute value components nor tuples with duplicate attribute value components are allowed. Valid time, however, is represented by a set-valued time-stamp that is associated with individual attributes. A time-stamp represents possibly disjoint intervals and the time-stamps assigned to two attributes in a given tuple may, but need not, be identical.

from having other, more highly desirable properties.

In the next section we define the historical algebra and provide a formal semantics. To do so, we redefine a *relation*, the only type of object allowed in the algebra, to include valid time. We also redefine the algebraic operators, and introduce new operators, to handle this new temporal dimension. We then show that the algebra has the expressive power of the *TQuel* (*T*emporal *QUE*ry *L*anguage) [Snodgrass 1987] facilities that support valid time. We demonstrate that several important properties: closure, relational completeness, and a restricted form of reducibility, as well as all but one of the traditional tautologies used in query optimization, hold for the algebra. Various implementation aspects, including page structure and incremental evaluation techniques, are considered. In particular, we show how the algebra may utilize a page layout that is quite similiar to that used by conventional DBMS's, and how the algebra can be efficiently implemented. We conclude with a discussion of the major design decisions we made in defining the algebra and the importance of those decisions in determining the algebra's properties. The notational conventions used in the paper are described in Appendix A.

2 Historical Relations

The algebra presented in this paper is an extension of the snapshot algebra. As such, it retains the basic restrictions on attribute values found in the snapshot algebra. Neither set-valued attributes nor tuples with duplicate attribute values are allowed. Valid time is represented by a set-valued time-stamp that is associated with individual attributes. A time-stamp represents possibly disjoint intervals and the time-stamps assigned to two attributes in a given tuple need not be identical.

Assume that we are given a relation scheme defined as a finite set of attribute names $\mathcal{N} = \{N_1, \ldots, N_m\}$. Corresponding to each attribute name $N \in \mathcal{N}$ is a domain $Dom(N)$, an arbitrary, non-empty, finite or denumerable set [Maier 83]. Let the positive integers be the domain $\mathcal{T}$, where each element of $\mathcal{T}$ represents a time quantum [Anderson 1982]. Assume that, if t_1 immediately precedes t_2 in the linear ordering of $\mathcal{T}$, then t_1 represents the interval $[t_1,\ t_2)$. The granularity of time (e.g., nanosecond, month, year) associated with $\mathcal{T}$ is arbitrary. Note that when we speak of a "point in time," we actually refer to an interval, termed a *chronon*, whose duration is determined by the granularity of the measure of time being used to specify that point in time. Also, let the domain $\wp(\mathcal{T})$ be the power set of $\mathcal{T}$. An element of $\wp(\mathcal{T})$ is then a set of integers, each of which represents an interval of unit duration. Also, any group of consecutive integers $t_1,\ \ldots,\ t_n$ appearing in an element of $\wp(\mathcal{T})$, together represent the interval $[t_1,\ t_n + 1)$. An element of $\wp(\mathcal{T})$, termed a *temporal element* [Gadia 1988], is thus a finite union of intervals.

If we let *value* range over the domain $Dom(N_1) \cup \cdots \cup Dom(N_m)$ and *valid* range over the domain $\wp(\mathcal{T})$, we can define an *historical tuple ht* as a mapping from the set of attribute names to the set of ordered pairs (*value*, *valid*), with the following restrictions.

- $\forall a,\ 1 \leq a \leq m\ (value(ht(N_a)) \in Dom(N_a)$ and
- $\exists a,\ 1 \leq a \leq m\ (valid(ht(N_a)) \neq \emptyset)$.

1 Introduction

Conventional database management systems do not support the time-varying aspects of the real world. In this paper we propose extending the relational algebra [Codd 1970] to enable it to handle time. Several benefits accrue from defining a historical algebra. A historical algebra is essential to the formulation of a historical data model because it defines formally the types of objects and the operations on object instances allowed in the data model. The usefulness of a historical data model in representing the time-varying aspect of real-world phenomena depends on the power and expressiveness of its underlying historical algebra. Similarly, the algebra determines a data model's support of calculus-based query languages. Also, implementation issues, such as query optimization and physical storage strategies, can best be addressed in terms of the algebra.

Conventional databases can be viewed as *snapshot* databases in that they represent the state of the real world at one particular point in time. As a database is changed to reflect changes in the real world, out-of-date information, representing past states of the real world, is deleted. In previous papers, we identified three orthogonal kinds of time that a database management system (DBMS) needs to support: valid time, transaction time, and user-defined time [Snodgrass & Ahn 1985, Snodgrass & Ahn 1986]. *Valid time* concerns modeling time-varying reality. The valid time of, say, an event is the clock time at which the event occurred in the real world, independent of the recording of that event in some database. *Transaction time*, on the other hand, concerns the storage of information in the database. The transaction time of an event is the transaction number (an integer) of the transaction that stored the information about the event in the database. *User-defined time* is an uninterpreted domain for which the DBMS supports the operations of input, output, and perhaps comparison. These three types of time are orthogonal in the support required of the DBMS.

The relational algebra already supports user-defined time in that user-defined time is simply another domain, such as integer or character string, provided by the DBMS [Bontempo 1983, Overmyer & Stonebraker 1982, Tandem 1983]. The relational algebra, however, supports neither valid time nor transaction time. Hence, for clarity, we refer to the relational algebra hereafter as the *snapshot* algebra and our proposed algebra, which supports valid time, as an *historical* algebra. We do not consider here extensions to support transaction time. Elsewhere we described an approach for adding transaction time to the snapshot algebra and showed that this approach applies without change to all historical algebras supporting valid time [McKenzie & Snodgrass 1987]. That approach also supports scheme evolution [McKenzie & Snodgrass 1990]. We provide formal definitions for historical relations and for their associated algebraic operators. The result is an algebra supporting all three kinds of time.

The algebra reflects our basic design goal to define an historical algebra that has as many desirable properties as possible. For example, we wanted the historical algebra to be a straightforward extension of the snapshot algebra so that relations and algebraic expressions in the snapshot algebra would have equivalent counterparts in the historical algebra. Yet we also wanted the algebra to support historical queries and adhere to the user-oriented model of historical relations as space-filling objects, where the additional, third dimension is valid time. Hence, we did not restrict historical relations to first normal form, insist on time-stamping of entire tuples, or require that time-stamps be atomic-valued, because each of these restrictions would have prevented the algebra

4.3.4 The Outer Where Clause 42
4.3.5 The Outer When Clause 42
4.3.6 Nested Aggregation 43
4.4 Correspondence Theorems 43

5 Implementing the Algebra **44**
5.1 Query Optimization 44
5.2 Page Structure 48
5.3 Incremental Execution 49

6 Review of Design Decisions **50**
6.1 Time-stamping Attribute Values 51
6.2 Set-valued Time-stamps 51
6.3 Single-valued Attributes 52
6.4 Extended Operator Semantics 52
6.5 New Temporal Operators 53

7 Summary **53**

8 Acknowledgements **54**

Bibliography **54**

A Notational Conventions **60**

B Auxiliary Functions **62**

Contents

1 Introduction **1**

2 Historical Relations **2**

3 Historical Operators **4**

3.1 Historical Analogues of the Snapshot Operators 4

3.2 Historical Derivation 7

3.3 Conversion Operators 11

3.4 Aggregates 11

3.4.1 Partitioning Function 14

3.4.2 Non-unique Aggregates 15

3.4.3 Unique Aggregates 16

3.4.4 Expressions in Aggregates 17

3.5 Closure, Completeness, and Reducibility 17

3.6 Additional Aspects of the Algebra 20

3.7 Summary 24

4 Equivalence with TQuel **25**

4.1 TQuel Retrieve Statement 25

4.2 Correspondence with the Historical Algebra 27

4.3 TQuel Aggregates 33

4.3.1 TQuel Aggregates in the Target List 37

4.3.2 The Inner Where Clause 40

4.3.3 The Inner When Clause 41