Extending the Relational Algebra to Support Transaction Time

Edwin McKenzie and Richard Snodgrass†*

Department of Computer Science
University of North Carolina
Chapel Hill, NC 27514

Abstract

In this paper we discuss extensions to the conventional relational algebra to support transaction time We show that these extensions are applicable to historical algebras that support valid time, yielding a temporal algebraic language Since transaction time concerns the storage of information in the database, the notion of state is central The extensions are formalized using denotational semantics The additions preserve the useful properties of the conventional relational algebra

1 Introduction

Codd's relational algebra [Codd 1970] is truly timeless, in several senses First, the relations it operates on model the current reality as is currently best known, the information approximates an instantaneous snapshot Secondly, while the computation of a relational algebraic expression occurs in an innermost-out fashion, there is no sense of the computation requiring time to complete Third, the disposition of the derived relation computed by the algebraic expression is ethereal, presumably this relation will be displayed or stored back in the database—the algebra will never tell

In this paper we propose extensions that address the first and third aspects Time must be added to the underlying data model before it can be added to the relational algebra 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 that 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 concerning 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 and minimal computation As its name implies, the semantics of user-defined time is provided by the user or application program These three types of time are orthogonal in the support required of the DBMS

In these same papers, we defined four classes of relational databases depending on their support for valid time and transaction time snapshot databases, rollback databases, historical databases, and temporal databases User-defined time is in fact already supported by the relational algebra, in that it is simply another domain, such as integer or character string, provided by the DBMS [Bontempo 1983, Overmyer & Stonebraker 1982, Tandem 1983] *Snap-*

*Research by this author was supported in part by the United States Air Force

†Research by this author was supported in part by an IBM Faculty Development Award

This research was also supported by NSF grant DCR-8402339

shot databases support neither valid time nor transaction time They represent a relation as a single snapshot state (i e , the state of the enterprise being modeled at one particular point in time) Snapshot databases are exactly those databases supported by the relational algebra Hence, for clarity, we will refer to the relational algebra hereafter as the snapshot algebra *Rollback databases* support transaction time but do not support valid time They represent a relation as a sequence of snapshot states indexed by transaction time By recording the history of database activity, rollback databases allow relations to be rolled back to one of their past snapshot states for querying *Historical databases* support valid time but do not support transaction time They represent a relation as a single historical state (i e , the history as is best known of the enterprise being modeled) By recording the history of the real world, historical databases provide support for historical queries When an historical database is changed, however, past historical states are not retained *Temporal databases* support both valid time and transaction time They represent a relation as a sequence of historical states indexed by transaction time By recording both the history of the enterprise being modeled and the history of database activities, temporal databases provide support for both historical queries and rollback operations

In this paper we discuss extensions to the snapshot algebra to enable it to handle transaction time There have already been several proposals for adding valid time to the algebra [Ben-Zvi 1982, Clifford & Croker 1987, Gadia 1984, Gadia 1986, Jones et al 1979, McKenzie & Snodgrass 1987B, Navathe & Ahmed 1986, Tansel 1986], so we will not consider extensions to support valid time Fortunately, since the two types of time are orthogonal, they can be studied in isolation We examine how transaction time can be added to the snapshot algebra and show how our approach applies without modification to all historical algebras supporting valid time, yielding a temporal algebraic language that can accommodate all three kinds of time

Several benefits accrue from extending the snapshot algebra to support transaction time The action of update is available in the algebra, allowing the algebra to be the executable form to which update operations in a calculus-based language (e g , append, delete, replace in Quel [Held et al 1975]) can be mapped If these operations in the calculus are formalized, the mapping can be proven correct Secondly, update optimizations analogous to the retrieval optimizations that have been extensively studied [Smith & Chang 1975] can now be investigated in a rigorous fashion. A third benefit is that the *contents* of the database, and its evolution, are now placed on a formal basis In particular, the domain of database states and the change to each state effected by each operator are defined Of course, actual implementations will vary considerably in the physical structures used to encode the information on secondary storage However, the existence of a formal definition of database state allows rigorous statements to be made concerning the correctness of those structures and the information content of the database

Additional benefits accrue from our approach for adding transaction time to the snapshot algebra First, our approach is general, it can be applied to any historical algebra to yield a temporal algebraic language Our approach for adding transaction time to the snapshot algebra depends on no specific technique for adding valid time to the snapshot algebra Rather, it is compatible with any such technique Secondly, our approach is consistent with the concepts of time-stamped concurrency control presented elsewhere [Bernstein et al 1987, Reed 1983, Rosenkrantz et al 1978, Stearns et al 1976]

2 The Approach

In adding transaction time to the relational model, we discovered a fundamental problem, that of *state* An algebra by definition is side-effect-free, but the essential aspect of a database transaction is solely its side-effect of modifying the database One awkward but perhaps feasible solution is to add the database as a parameter to every operator We adopt a different strategy, leaving the basic structure of the algebra intact, and instead inserting it into another structure of *commands* that provide the needed side-effects Hence, what we are proposing in this paper is not only an extended algebra, but a *language* with the (slightly extended) algebra as a significant component In doing so, we preserve all the properties of the snapshot algebra (e g , commutativity of select, distributivity of select over join), permitting the full application of previously developed algebraic optimizations

We employ denotational semantics to define the semantics of commands, due to its success in formalizing operations involving side-effects, such as assignment, in programming languages [Gordon 1979, Stoy 1977] The language thus defined is our proposal for adding transaction time to the relational model in

order to support a *rollback* relation as a sequence of snapshot states indexed by transaction time It is consistent with Maier's definition of a snapshot state and the snapshot algebra [Maier 1983]

A second modification does involve an extension to the snapshot algebra When transaction time is supported by a DBMS, a means of accessing states other than the current one must be included We define a new algebraic operator called *rollback* to make past states available in the algebra Fortunately, rollback is side-effect-free, so it is easily incorporated into the algebra

Valid time is supported by allowing a relation to contain one or more *historical states* Each historical state models the history of changes in the real world An *historical* relation contains a single historical state, and models the history as is currently best known A *temporal* relation contains a sequence of historical states, each modeling the history as it was stored in the database at a particular point in time Our language is consistent with definitions of historical state and historical relational algebras proposed by others [Clifford & Croker 1987, Gadia 1984, Gadia 1986, Jones et al 1979, McKenzie & Snodgrass 1987B, Navathe & Ahmed 1986, Tansel 1986]

In defining the semantics of commands and algebraic operators, we have favored simplicity of semantics at the expense of efficient direct implementation The language would be quite inefficient, in terms of storage space and execution time, if mapped directly into an implementation However, the semantics do not preclude more efficient implementations using optimization strategies for both storage and retrieval of information

Summarizing the changes, we add

- commands formalized using denotational semantics to express additions to the state of the database,
- a rollback operator to the algebra to access previous states, and
- valid time, accommodated by permitting historical states to be stored in relations

The first two changes will be the topic of the next section Section 4 will address incorporating valid time and section 5 will compare our approach with those of others

3 Commands and the Rollback Operator

In denotational semantics, a language is described by assigning to each language construct a *denotation* – an abstract entity which models its meaning We chose denotational semantics as the methodology for defining our language because denotational semantics combines a powerful descriptive notation with rigorous mathematical theory to allow the precise definition of state First, we define the syntax of our language Then we define the semantic domains of the language and several auxiliary functions Finally, we define the semantic functions which map the language constructs into their denotations

3.1 Syntax

Our language has three basic types of language constructs sentences, commands, and expressions A sentence in our language is a non-empty sequence of commands Commands are analogous to statements in Quel or SQL in that they specify some task that either queries or changes the database (e g , define a relation, modify the contents of a relation, display the contents of a relation) Expressions occur within commands and always evaluate to a single snapshot state We represent these three types of constructs by the three syntactic domains

EXPRESSION	Domain of expressions
COMMAND	Domain of commands
SENTENCE	Domain of sentences

We use Backus-Naur Form to specify here the syntax of expressions and commands in terms of their *immediate constituents* (i e , the highest-level constructs which make up expressions and commands) The complete syntax of the language, including definitions of the lower-level constituents such as identifiers, snapshot states, and boolean expressions, is given elsewhere [McKenzie & Snodgrass 1987A] If we let

E, E_1, and E_2 range over the domain *EXPRESSION*,

C, C_1, and C_2 range over the domain *COMMAND*,

P range over the domain *SENTENCE*,

I range over the domain *IDENTIFIER* of

alphanumeric identifiers,

N range over the domain $\mathcal{NUMERAL}$ of decimal numerals and the special symbol ∞,

S range over the domain $\mathcal{STRING}$ of strings in an alphabet,

A range over the domain $\mathcal{STATE}$ of alphanumeric representations of snapshot states (i e , constant relations),

Y range over the domain $\mathcal{TYPE}$ of character strings denoting relation types (i e , `snapshot`, `rollback`),

X range over the domain $\mathcal{P}(\mathcal{IDENTIFIER})$, the power set of $\mathcal{IDENTIFIER}$, and

F range over the domain $\mathcal{F}$ of boolean expressions of elements from the domains $\mathcal{IDENTIFIER}$ and $\mathcal{STRING}$, the relational operators, and the logical operators

then the syntax for the language is

$$\begin{aligned} E &= A \mid E_1 \cup E_2 \mid E_1 - E_2 \mid E_1 \times E_2 \\ &\quad \mid \pi_X(E) \mid \sigma_F(E) \mid \rho(I, N) \\ C &= \texttt{define\_relation}(I, Y) \\ &\quad \mid \texttt{modify\_state}(I, E) \mid C_1, C_2 \\ P &= C \end{aligned}$$

An expression may be a snapshot state or an algebraic operator on either one or two other expressions The allowable operators include the five operators that serve to define the snapshot algebra To these, we have added an additional operator, a rollback operator ρ The rollback operator takes two arguments, the name of a relation (an $\mathcal{IDENTIFIER}$) and a transaction number (a $\mathcal{NUMERAL}$), and retrieves the snapshot state from the named relation current at the time of the indicated transaction

There are two commands in the language The `define_relation` command binds a relation type Y and an empty sequence of snapshot states to an unbound identifier I The `modify_state` command changes a relation's state but leaves the relation's type unchanged The command evaluates an expression E to produce a snapshot state which becomes the current state of relation I This new state may, but need not, contain tuples from the relation's previous state as well as tuples not found in the relation's previous state Tuples from the relation's previous state may appear unchanged or have different values for some attributes in the new state Thus, the `modify_state` command effectively performs append, delete, and replace operations (e g , Quel [Held et al 1975]) on relations by adding tuples to, deleting tuples from, or replacing tuples in a relation's previous state to produce a new current state for that relation. For simplicity, we assume that there is no `delete_relation` command in the language A relation, once defined, cannot be deleted A relation's state may be changed, but the relation itself exists permanently (We assume that the database administrator will have additional facilities to migrate rollback relations to tape) Elsewhere we introduce into the language a `delete_relation` command, applicable to both snapshot and rollback relations [McKenzie & Snodgrass 1987A]

How changes to a relation's state are handled depends on the relation's type For snapshot relations, a state change causes the most recent state in the relation's sequence of states to be replaced by the new state For rollback relations, a state change causes the new state to be concatenated at the end of the relation's sequence of states Thus, while only the most recent state of snapshot relations is saved, all past states of rollback relations are saved Note that the sequence of states for a snapshot relation will always be a single-element sequence

The rollback operator ρ retrieves the state of relation I at the time of transaction N The behavior of this operator depends on whether or not the argument N is ∞ If N is ∞, ρ retrieves the state of a relation at the time of the most recent transaction on the database In this case, the operator ρ may be applied to either a snapshot or a rollback relation, retrieving the relation's most recent state If N is not ∞, ρ may only be applied to a rollback relation Thus, the rollback operator retrieves either the current state of a snapshot or rollback relation or a past state of a rollback relation The rollback operator cannot retrieve a past state of a snapshot relation

3.2 Semantic Domains

$\mathcal{SENTENCE}$ is the set of all syntactically valid sentences in our language Each sentence, which consists of a sequence of one or more individual commands, defines the database resulting from the execution of those commands, in order, on an empty database As we will see later, the syntactic domain

of sentences is needed only to ensure this restriction By defining the database that results from an arbitrary sentence, we specify the semantics of that sentence, and hence the semantics of the language In this section, we will formally define the domain of database states, subsequent sections will provide the connection between the syntactic domain of sentences and the semantic domain of database states

Assume that we are given a set of domains $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_m\}$, where each domain $\mathcal{D}_i$, $1 \leq i \leq m$, is an arbitrary, non-empty, finite or countably infinite set Then, we can define the following semantic domains for our language

$$\mathcal{TRANSACTION\ NUMBER} \triangleq \{0, 1, \ldots\}$$

A transaction number is a non-negative integer which is used to identify a transaction that modifies the database The transaction number assigned to a transaction can be viewed as that transaction's time-stamp We assume that database modifications occur sequentially and that a transaction's time-stamp as represented by its transaction number is the *commit* time for the transaction (i e , the time the database is actually changed as a result of the transaction's execution) We note in passing that implementations may use some other time, such as the *begin transaction* time for the transaction, for greater efficiency (e g , POSTGRES [Stonebraker & Rowe 1986]) However, such implementations should preserve the semantics of commit transaction time as specified here Implementations may also permit concurrent transactions, again as long as the semantics of sequential update with a monotonically increasing transaction time is preserved

$$\mathcal{RELATION\ TYPE} \triangleq \{\text{SNAPSHOT, ROLLBACK}\}$$

$\mathcal{SNAPSHOT\ STATE} \triangleq$ Domain of all valid snapshot states, as defined in the snapshot algebra [Maier 1983], over elements of $\{\mathcal{D}_1 \cup \mathcal{D}_2 \cup \ldots \cup \mathcal{D}_m\}$

$$\mathcal{RELATION} \triangleq \mathcal{RELATION\ TYPE} \times [\mathcal{SNAPSHOT\ STATE} \times \mathcal{TRANSACTION\ NUMBER}]^*$$

A relation is an ordered pair consisting of

- a relation type, and
- a sequence of (snapshot state, transaction number) pairs

A relation's state is dynamic When a transaction changes the state of a snapshot relation, the single element in the relation's state sequence is replaced by a new element consisting of a new snapshot state and associated transaction number When a transaction changes the state of a rollback relation, a new pair consisting of a new snapshot state and associated transaction number is appended to the relation's existing state sequence Thus, rollback relations are append only relations defined in terms of snapshot states

Note that the transaction number of each element in a relation's state sequence can be viewed as a time-stamp indicating when its associated relation state was entered into the database and became the relation's current state Since we assume that database modifications occur sequentially, the transaction-number components of a state sequence, while not necessarily consecutive, will be nevertheless strictly increasing (as a consequence of transaction time being associated with commit) Thus, we can interpolate on the transaction-number component of elements in a given state sequence to determine the state of a rollback relation at any time

$$\mathcal{DATABASE\ STATE} \triangleq \mathcal{IDENTIFIER} \rightarrow [\mathcal{RELATION} + \{\perp\}]$$

A database state is a function that maps identifiers either into a relation or into the special symbol $\perp$, which here indicates that the identifier is unbounded in that database state (i e , is associated with no relation) The notation "+" on domains means the disjoint union of domains

$$\mathcal{DATABASE} \triangleq \mathcal{DATABASE\ STATE} \times \mathcal{TRANSACTION\ NUMBER}$$

A database is an ordered pair consisting of a database state and a transaction number indicating the most recent transaction that caused a change to the database

3.3 Auxiliary Functions

To specify the meaning of the expressions and commands defined syntactically in Section 3 1, we will define a function mapping each valid expression into a snapshot state (i e , an element of the $\mathcal{SNAPSHOT\ STATE}$ semantic domain) and a function mapping each valid command into a database (i e , an element of the $\mathcal{DATABASE}$ semantic domain) We use

several auxiliary functions in the definitions of these semantic functions for expressions and commands We present here an informal description of each of these auxiliary functions A formal definition for **FINDSTATE** is presented elsewhere [McKenzie & Snodgrass 1987A] Formal definitions for the other functions are either straightforward or notationally cumbersome and hence are not presented

RTYPE maps a relation into its relation type

RSTATE maps a relation into its sequence of (snapshot state, transaction number) pairs

FINDSTATE maps a relation into the snapshot-state component of the element in the relation's state sequence having the largest transaction-number component less than or equal to a given integer If the sequence is empty or no such element exists in the sequence, then **FINDSTATE** returns the empty set

N is a semantic function which maps the syntactic domain *NUMERAL* of decimal numerals into the semantic domain *TRANSACTION NUMBER* of non-negative integer numbers

Y is a semantic function which maps each character string in the syntactic domain *TYPE* into the relation type which it denotes

S is a semantic function which maps each alphanumeric representation of a snapshot state in the syntactic domain *STATE* into its corresponding snapshot state in the semantic domain *SNAPSHOT STATE*

3.4 Expressions

We now define the semantic function **E**, which defines the denotation of expressions in our language, as follows

$$\mathbf{E}\ \mathit{EXPRESSION} \rightarrow [\mathit{DATABASE} \rightarrow [\mathit{SNAPSHOT\ STATE}]]$$

The result of evaluating an expression on a specific database is a snapshot state Note that evaluation of an expression on a specific database does not change that database

This definition of the semantic function **E** does not handle the possibility that an expression, when evaluated on a specific database, causes an error (e g , an attempt to project a non-existent attribute) We limit our discussion of expressions to valid expressions on a given database Thus, the semantic function **E**, which defines the denotation of expressions in our language, is a partial function on valid expressions only A discussion of invalid expressions and a mechanism for handling such expressions appears elsewhere [McKenzie & Snodgrass 1987A]

We now formally define the semantic function **E** for each kind of expression allowed in the language If we let

n range over the domain *TRANSACTION NUMBER*,
r range over the domain *RELATION*,
b range over the domain *DATABASE STATE*, and
d range over the domain *DATABASE*,

then

$$\mathbf{E}[\![A]\!]\,d \triangleq \mathbf{S}[\![A]\!]$$

$$\mathbf{E}[\![E_1 \cup E_2]\!]\,d \triangleq \mathbf{E}[\![E_1]\!]\,d \cup \mathbf{E}[\![E_2]\!]\,d$$

$$\mathbf{E}[\![E_1 - E_2]\!]\,d \triangleq \mathbf{E}[\![E_1]\!]\,d - \mathbf{E}[\![E_2]\!]\,d$$

$$\mathbf{E}[\![E_1 \times E_2]\!]\,d \triangleq \mathbf{E}[\![E_1]\!]\,d \times \mathbf{E}[\![E_2]\!]\,d$$

$$\mathbf{E}[\![\pi_X(E)]\!]\,d \triangleq \pi_X(\mathbf{E}[\![E]\!]\,d)$$

$$\mathbf{E}[\![\sigma_F(E)]\!]\,d \triangleq \sigma_F(\mathbf{E}[\![E]\!]\,d)$$

We now define the semantic function **E** for the new rollback operator ρ

$$\mathbf{E}[\![\rho(I, N)]\!]\,d \triangleq \text{if } N = \infty \text{ then } \mathbf{FINDSTATE}(r, n) \text{ else } \mathbf{FINDSTATE}(r, \mathbf{N}[\![N]\!])$$

where $d = (b, n)$ and $r = b(I)$ If $N = \infty$, then the result of evaluating the expression $\rho(I, N)$ is the most recent snapshot state in the state sequence of the relation corresponding to the identifier I If $N \neq \infty$, then the result of evaluating the expression $\rho(I, N)$ is the snapshot state associated with the largest transaction number less than or equal to the transaction number $\mathbf{N}[\![N]\!]$ in the state sequence of the relation corresponding to the identifier I Thus, the operator ρ either retrieves the current state of the relation identified by I or rolls back the relation to its state at the time the transaction associated with $\mathbf{N}[\![N]\!]$ was processed This definition assumes if $N = \infty$ that the relation is either a snapshot or rollback relation and if $N \neq \infty$ that the relation is

a rollback relation, otherwise, the rollback operation would be illegal

3.5 Commands

Commands are the only language constructs that change the database Execution of a command either produces a new database or leaves the database unchanged

$$\mathbf{C}\ \ COMMAND \rightarrow [DATABASE \rightarrow [DATABASE]]$$

We define formally the semantics of commands using the same approach we used to define the semantics of expressions We define the semantic function C for each kind of command allowed in the language

The command `define_relation` defines a new, active relation in the database

$$\begin{aligned}&\mathbf{C}[\![\texttt{define\_relation}(I,\ Y)]\!]\,d \triangleq\\ &\qquad \text{if} \quad b(I) = \perp\\ &\qquad \text{then} \quad (b\,[(\mathbf{Y}[\![Y]\!],\ \langle\,\rangle)/I\,],\ n+1)\\ &\qquad \text{else} \quad d\end{aligned}$$

where $d = (b, n)$ If the database's database-state component maps the identifier I into $\perp$, then the command `define_relation`$(I,\ Y)$ changes the database so that the database's database-state component maps the identifier I into an empty sequence of relation states of relation type $\mathbf{Y}[\![Y]\!]$ The transaction number for the database is also incremented If the database's database-state component does not currently map the identifier I into $\perp$, then the identifier already denotes a defined relation and the command leaves the database unchanged

The command `modify_state` either replaces the single element in the state sequence of a defined snapshot relation or adds a new element to the state sequence of a defined rollback relation

$$\begin{aligned}&\mathbf{C}[\![\texttt{modify\_state}(I,\ E)]\!]\,d \triangleq\\ &\quad \text{if} \quad r \neq \perp \wedge \mathbf{RTYPE}\,(r) = \text{SNAPSHOT}\\ &\quad \text{then} \quad (b[(\mathbf{RTYPE}(r),\\ &\qquad \langle(\mathbf{E}[\![E]\!]\,d,\ n+1)\rangle)/I],\ n+1)\\ &\quad \text{else if} \quad r \neq \perp \wedge \mathbf{RTYPE}\,(r) = \text{ROLLBACK}\\ &\quad \text{then} \quad (b[(\mathbf{RTYPE}(r),\\ &\qquad \mathbf{RSTATE}(r) \,\|\, (\mathbf{E}[\![E]\!]\,d,\ n+1))/I],\\ &\qquad n+1)\\ &\quad \text{else} \quad d\end{aligned}$$

where $d = (b, n)$, $r = b(I)$, and "$\|$" is the concatenation operator on sequences If the database's database-state component maps the identifier I into a defined snapshot relation, then the `modify_state` command replaces the relation with a new relation consisting of its type ($\mathbf{RTYPE}(r)$) and a new state sequence This state sequence is a single-element sequence consisting of the new (snapshot state, transaction number) pair $(\mathbf{E}[\![E]\!]\,d,\ n+1)$ If the database's database-state component maps the identifier I into a defined rollback relation, then the `modify_state` command replaces the relation with a new relation consisting of its type and its state sequence to which is concatenated at the end a new (snapshot state, transaction number) pair Hence, the single state in snapshot relations is *replaced* with the state resulting from the evaluation of E, whereas, a new state is *appended* in rollback relations In either case, the new (snapshot state, transaction number) pair is the snapshot state $\mathbf{E}[\![E]\!]\,d$ and the transaction number of the most recent transaction on the database plus one The `modify_state` command supports all update operations Append is accommodated by an expression E that evaluates to a snapshot state containing all of the tuples in a relation's most recent state plus one or more tuples not in the relation's most recent state Delete is accommodated by an expression E that evaluates to a snapshot state containing only a proper subset of the tuples in a relation's most recent state Finally, replace is accommodated by an expression E that evaluates to a snapshot state that differs from a relation's most recent state only in the attribute values of one or more tuples

If two commands appear in sequence, command C_1 is executed first Then, command C_2 is executed using the database resulting from the execution of command C_1

$$\mathbf{C}[\![C_1,\ C_2]\!]\,d \triangleq \mathbf{C}[\![C_2]\!](\mathbf{C}[\![C_1]\!]\,d)$$

3.6 Sentences

Sentences are the highest-level construct in our language A sentence defines the database state resulting from the execution of a sequence of one or more commands, starting with the empty database Our language requires that the evaluation of a sentence in the language always start with an empty database This requirement is both necessary and sufficient, given the above definitions of the commands `define_relation` and `modify_state`, to ensure that transaction-number components of the state se-

quence of each rollback relation in the database will be strictly increasing. The content of a database is the cumulative result of all the transactions that have been performed on it since it was created

$$\mathbf{P} \quad \mathit{SENTENCE} \rightarrow [\mathit{DATABASE}]$$

$$\mathbf{P}[\![C]\!] \triangleq \mathbf{C}[\![C]\!](\mathbf{EMPTY}, 0)$$

where $\mathbf{EMPTY} \quad \mathit{IDENTIFIER} \rightarrow \{\perp\}$ The database-state component of the database is defined to be the function which maps all identifiers to $\perp$ (i e , no identifier is associated with a relation) and the transaction-count component of the database is set to 0

4 Supporting Both Valid Time and Transaction Time

The previous section showed how the snapshot algebra can be extended to handle transaction time by defining a rollback operator and several commands that modify the database Since valid time and transaction time are orthogonal concepts, it is possible to extend an historical algebra in much the same way to obtain a temporal algebraic language We now show how to extend an historical algebra to support transaction time For illustration, we will use one particular historical algebra (defined elsewhere [McKenzie & Snodgrass 1987B]), but the approach applies to any historical algebra

The key aspect of an historical algebra is its definition of historical state, which models reality over a period of time By storing an historical state, this model can be captured for further analysis An historical relation will consist of exactly one historical state A temporal relation will contain a sequence of historical states, indexed by transaction time, a new rollback operator $\hat{\rho}$ will be used to access a particular historical state

We first define the syntax of the historical algebra by redefining two syntactic domains and introducing two additional syntactic domains If we let

A range over the domain STATE of alphanumeric representations of snapshot and historical states,

Y range over the domain TYPE of character strings denoting relation types (i e , **snapshot**, **rollback**, **historical**, **temporal**),

V range over the domain $\mathcal{V}$ of temporal expressions, and

G range over the domain $\mathcal{G}$ of boolean expressions of elements from the domain $\mathcal{V}$, the relational operators, and the logical operators

then the syntax for the language may be extended with

$$\begin{aligned} E \quad = \quad & (Y, A) \mid E_1 \hat{\cup} E_2 \mid E_1 \hat{-} E_2 \mid E_1 \hat{\times} E_2 \\ & \mid \hat{\pi}_X(E) \mid \hat{\sigma}_F(E) \mid \delta_{G,V}(E) \mid \hat{\rho}(I, N) \end{aligned}$$

The constant may now be a snapshot or historical state and is extended to specify the relation type The first five operators are historical counterparts to conventional algebraic operators Each is represented as $\hat{op}$ to distinguish it from its snapshot algebra counterpart op The sixth operator $\delta_{G,V}$ is a new historical operator which performs functions, similar to those of the selection and projection operators in the snapshot algebra, on the valid-time components of historical tuples The seventh operator is an historical counterpart of the rollback operator defined on temporal relations All evaluate to historical states

Next we extend the set of relation types, define the domain of historical states, and augment the definition of the $\mathit{RELATION}$ domain

$$\mathit{RELATION\ TYPE} \triangleq \{\text{SNAPSHOT, ROLLBACK, HISTORICAL, TEMPORAL}\}$$

$\mathit{HISTORICAL\ STATE} \triangleq$ Domain of all valid historical relations as defined in the historical algebra

$$\begin{aligned} \mathit{RELATION} \triangleq\ & \mathit{RELATION\ TYPE} \times \\ & [[\mathit{SNAPSHOT\ STATE} \times \\ & \quad \mathit{TRANSACTION\ NUMBER}]^* + \\ & \quad [\mathit{HISTORICAL\ STATE} \times \\ & \quad \mathit{TRANSACTION\ NUMBER}]^*] \end{aligned}$$

We also need one more semantic function, **H**, which maps an alphanumeric representation of an historical state in the syntactic domain STATE into its corresponding historical state in the semantic domain $\mathit{HISTORICAL\ STATE}$

Definitions of the semantic function **E** for expressions involving historical operators are specified next The denotations for this class of expres-

sions are analogous to those for expressions involving snapshot operators

$$\mathbf{E}[\![(Y, A)]\!]d \triangleq \text{if } \mathbf{Y}[\![Y]\!] = \text{SNAPSHOT} \text{ then } \mathbf{S}[\![A]\!] \text{ else } \mathbf{H}[\![A]\!]$$

$$\mathbf{E}[\![E_1 \hat{\cup} E_2]\!]d \triangleq \mathbf{E}[\![E_1]\!]d \hat{\cup} \mathbf{E}[\![E_2]\!]d$$

$$\mathbf{E}[\![E_1 \hat{-} E_2]\!]d \triangleq \mathbf{E}[\![E_1]\!]d \hat{-} \mathbf{E}[\![E_2]\!]d$$

$$\mathbf{E}[\![E_1 \hat{\times} E_2]\!]d \triangleq \mathbf{E}[\![E_1]\!]d \hat{\times} \mathbf{E}[\![E_2]\!]d$$

$$\mathbf{E}[\![\hat{\pi}_X(E)]\!]d \triangleq \hat{\pi}_X(\mathbf{E}[\![E]\!]d)$$

$$\mathbf{E}[\![\hat{\sigma}_F(E)]\!]d \triangleq \hat{\sigma}_F(\mathbf{E}[\![E]\!]d)$$

$$\mathbf{E}[\![\delta_{G,V}(E)]\!]d \triangleq \delta_{G,V}(\mathbf{E}[\![E]\!]d)$$

$$\mathbf{E}[\![\hat{\rho}(I, N)]\!]d \triangleq \text{if } N = \infty \text{ then } \mathbf{FINDSTATE}(r, n) \text{ else } \mathbf{FINDSTATE}(r, \mathbf{N}[\![N]\!])$$

where $d = (b, n)$ and $r = b(I)$

Finally, the `modify_state` command must be extended slightly to handle the historical and temporal relation types

$$\begin{aligned}
&\mathbf{C}[\![\texttt{modify_state}(I, E)]\!]d \triangleq \\
&\quad \text{if} \quad r \neq \perp \wedge \\
&\qquad (\mathbf{FINDTYPE}(r, n) = \text{SNAPSHOT} \vee \\
&\qquad\ \mathbf{FINDTYPE}(r, n) = \text{HISTORICAL}) \\
&\quad \text{then} \quad (b[(\mathbf{RTYPE}(r), \\
&\qquad\ \langle(\mathbf{E}[\![E]\!]d, n+1)\rangle)/I], n+1) \\
&\quad \text{else if} \quad r \neq \perp \wedge \\
&\qquad (\mathbf{FINDTYPE}(r, n) = \text{ROLLBACK} \vee \\
&\qquad\ \mathbf{FINDTYPE}(r, n) = \text{TEMPORAL}) \\
&\quad \text{then} \quad (b[(\mathbf{RTYPE}(r), \\
&\qquad\ \mathbf{RSTATE}(r) \parallel (\mathbf{E}[\![E]\!]d, n+1))/I], \\
&\qquad\ n+1) \\
&\quad \text{else} \quad d
\end{aligned}$$

where $d = (b, n)$ and $r = b(I)$ Notice that historical relations are handled similarly to snapshot relations, the only difference is that **E** evaluates to a historical state rather than a snapshot state The same relationship holds between rollback and temporal relations By embedding the algebra in the structure of commands, we have emphasized the orthogonality of transaction and valid time Valid time is handled through new historical algebraic operators and a definition of historical state, transaction time is handled through the `modify_state` command and the rollback operator(s) In a sense, our semantics provides additional assurance that the two kinds of time are in fact orthogonal

5 Related Work and Summary

There are two contributions of this paper The first is that the database state is modeled as a sequence of snapshot (or historical) states indexed by transaction time This approach is similar to that proposed in the context of time-stamp concurrency control algorithms [Bernstein et al 1987, Reed 1983, Rosenkrantz et al 1978, Stearns et al 1976] and dynamic constraints [Vianu 1983]. In a related effort, Abiteboul and Vianu have defined a transaction language TL consisting of parameterized expressions containing tuple insertions and deletions and a looping construct [Abiteboul & Vianu 1987] In TL, the database state is modeled "procedurally" by providing the transaction(s) that compute that state, transaction time is implicit The focus of this and previous research [Abiteboul & Vianu 1985, Abiteboul & Vianu 1986, Vianu 1983] is developing a characterization of the possible database states computable by constrained transactions, with the goal of using such transactions as a specification tool for stating dynamic constraints The goal of our language is different, we hope to model the evolution of the database in terms of transactions specified by the user in a calculus-based update language that is translated by the DBMS into algebraic expressions

There has been one other attempt to incorporate both valid time and transaction time in an algebra [Ben-Zvi 1982] Valid time and transaction time were supported through the addition of implicit time attributes to each tuple in a relation The algebra was extended with the *Time-View* algebraic operator which takes a relation and two times as arguments and produces the subset of tuples in the relation valid at the first time (the valid time) as of the second time (the transaction time) The Time-View operator thus rolls back a relation to a transaction time but returns only a subset of the tuples in the relation at that transaction time (i e , those tuples valid at some specified time) This restricted definition of the Time-View operator is tied inextricably

to his particular handling of valid time Our approach is compatible with any historical algebra

The second contribution is the formalization of the evolving state through the definition of the **modify_state** command This aspect has been investigated at the conceptual level by several researchers in the context of dynamic constraints on updates of database instances [Brodie 1981, Ceri et al 1981, Hammer & McLeod 1981] At the logical level, only Ben-Zvi has attempted such a formalization His approach is to provide procedures for various manipulation commands (e g , insert, delete, terminate) and prove that these procedures maintain various desirable properties The effect of these procedures are localized to a specific tuple that changes during the transaction Our **modify_state** command simply replaces or appends a new entire snapshot or historical state, allowing many tuples to change during a transaction Of course, actual implementations would be based on more complex representations that exhibit greater space and time efficiency Verifying the correctness of such implementations would involve demonstrating the equivalence of their semantics with the simple semantics presented here

An aspect concerning transaction time that is not addressed in this paper is *scheme evolution* The scheme is associated solely with transaction time, since it defines how reality is modeled by the database For example, a person's marital status is a (time-varying) aspect of reality, but the decision as to whether to record marital status, encoded in the scheme, is a (time-varying) aspect of the database Hence, as the scheme describes how data are stored in the database, changes to the scheme are properly the province of transaction time Elsewhere we provide extensions to the language presented here to accommodate scheme evolution [McKenzie & Snodgrass 1987A] We include a **delete_relation** command as part of those extensions

Another aspect that requires further work is that of completeness One approach is to define a language and propose it as a standard, Codd proposed his snapshot algebra as the yardstick for snapshot completeness (i e , supporting neither transaction nor valid time) Several others have proposed notions of query completeness based on computability [Abiteboul & Vianu 1987, Chandra & Harel 1980], which, unfortunately, are incomparable We feel that this latter approach is preferable and await a consensus to form against which we could measure our language for rollback completeness (i e , supporting transaction time) Similar statements apply to historical and temporal completeness, supporting valid and both kinds of time respectively [Snodgrass 1987]

In summary, this paper has defined an algebraic language that has a simple semantics and handles valid, transaction, and user-defined time Only two additional operators, ρ and $\hat{\rho}$, were necessary The additions required for transaction time did not compromise any of the useful properties of the snapshot algebra

6 Bibliography

[Abiteboul & Vianu 1985] Abiteboul, S and V Vianu *Transactions and Integrity Constraints,* in *Proceedings of the ACM Symposium on Principles of Database Systems,* 1985, pp 193-204

[Abiteboul & Vianu 1986] Abiteboul, S and V Vianu *Deciding Properties of Transactional Schemas,* in *Proceedings of the ACM Symposium on Principles of Database Systems,* 1986, pp 235-239

[Abiteboul & Vianu 1987] Abiteboul, S and V Vianu *A Transaction Language Complete for Database Update and Specification,* in *Proceedings of the ACM Symposium on Principles of Database Systems,* San Diego, CA Mar 1987

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

[Bernstein et al 1987] Bernstein, P A , V Hadzilacos and N Goodman *Concurrency Control and Recovery in Database Systems* Addison-Wesley Series in Computer Science Addison-Wesley, 1987

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

[Brodie 1981] Brodie, M *On Modelling Behavioral Semantics of Databases,* in *Proceedings of the Conference on Very Large Databases,* Cannes, France Sep 1981, pp 32-42

[Ceri et al 1981] Ceri, S , G Pelagatti and G Bracchi *Structured Methodology for Designing Static and Dynamic Aspects of Data Base Applications Information Systems,* 6, No 1 (1981),

pp 31-45

[Chandra & Harel 1980] Chandra, A K and D Harel *Computable Queries for Relational Data Bases Journal of Computer and Systems Science,* 21, No 2, Oct 1980, pp 156-178

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

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

[Gadia 1984] Gadia, S K *A Homogeneous Relational Model and Query Languages for Temporal Databases* 1984 (Submitted for publication)

[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

[Gordon 1979] Gordon, Michael J C *The Denotational Description of Programming Languages* New York-Heidelberg-Berlin Springer-Verlag, 1979

[Hammer & McLeod 1981] Hammer, M and D McLeod *Database Description with SDM A Semantic Database Model ACM Transactions on Database Systems,* 6, No 3, Sep 1981, pp 351-386

[Held et al 1975] Held, G D , M Stonebraker and E Wong *INGRES-A Relational Data Base Management System Proceedings of the AFIPS 1975 National Computer Conference,* 44, May 1975, pp 409-416

[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

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

[McKenzie & Snodgrass 1987A] McKenzie, E and R Snodgrass *Scheme Evolution and the Relational Algebra* Technical Report TR87-003 Computer Science Department, University of North Carolina at Chapel Hill Mar 1987

[McKenzie & Snodgrass 1987B] McKenzie, E and R Snodgrass *Supporting Valid Time An Historical Algebra and Evaluation* Technical Report TR87-008 Computer Science Department, University of North Carolina at Chapel Hill Apr 1987

[Navathe & Ahmed 1986] Navathe, S B and R Ahmed *A Temporal Relational Model and a Query Language* UF-CIS Technical Report TR-85-16 Computer and Information Sciences Department, University of Florida Apr 1986

[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

[Reed 1983] Reed, D P *Implementing Atomic Actions on Decentralized Data ACM Transactions on Computer Systems,* 1, No 1, Feb 1983, pp 3-23

[Rosenkrantz et al 1978] Rosenkrantz, D J , R E Stearns and P M Lewis *System Level Concurrency Control for Distributed Database Systems ACM Transactions on Database Systems,* 3, No 2, June 1978, pp 178-198

[Smith & Chang 1975] Smith, J M and P Y-J 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 (to appear),* 12, No 2, June 1987

[Stearns et al 1976] Stearns, R E, P M Lewis and D J Rosenkrantz *Concurrency Control for Database Systems,* in *Proceedings of the 17th Symposium on Foundations of Computer Science,* IEEE 1976, pp 19-32

[Stonebraker & Rowe 1986] Stonebraker, M and L A Rowe *The Design of POSTGRES,* in *Proceedings of ACM SIGMOD International Conference on Management of Data,* Ed C Zaniolo Association for Computing Machinery Washington, DC May 1986, pp 340-355

[Stoy 1977] Stoy, Joseph E *Denotational Semantics The Scott-Strachey Approach to Programming Language Theory* The MIT Series in Computer Science The MIT Press, 1977

[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

[Vianu 1983] Vianu, V *Dynamic Constraints and Database Evolution,* in *Proceeding of the ACM SIGAct-SIGMod Symposium on Principles of Database Systems,* Association for Computing Machinery Atlanta, GA Mar. 1983, pp 389-399