INTERVAL QUERIES ON OBJECT HISTORIES: EXTENDED ABSTRACT

Seymour Ginsburg and Katsumi Tanaka

University of Southern California
Los Angeles, U.S.A.

Kobe University
Kobe, JAPAN

ABSTRACT

This extended abstract introduces the notion of 'interval queries' on historical data for objects (here, called 'object histories') and explores a certain closure property. As for describing historical data for objects, we use our mathematical model introduced in our earlier paper. The major construct in the model is a 'computation-tuple sequence scheme' (abbreviated CSS), which specifies the set of all possible 'valid' object histories for the same type of object. Interval queries are those queries which return an interval (history) from a given object history. We provide conditions under which an interval query applied to object histories described by one CSS yields, as its answers, the set of all object histories which can be described by another CSS.

INTRODUCTION

Recently, much attention has been focussed on management of 'historical' databases or temporal information, and their associated research problems [A, B, CW, K]. In our earlier paper [GT1], we introduced a simple tuple-sequence-based data model for describing historical data for 'objects' (here, called 'object histories'). A notable feature of our model is that ordering of tuples and computation functions are explicitly handled. The major construct in the model is a 'computation-tuple sequence scheme' (abbreviated CSS), which specifies the set of all possible 'valid' object histories for the same type of object. The main purpose of the extended abstract is to introduce the notion of 'interval queries' on object histories, which return an interval (history) from a given object history, and to explore a certain closure property. (For more details, see [GT2].)

Informally, an object history is a historical record of an object. (Here, each object stands for an individual 'thing' or 'entity', such as a specific person's checking account, a specific professor's seminar, etc.) An object history is viewed as a sequence of occurrences, each occurrence consisting of some input data and, possibly, some calculation. In our model, each object history is represented simply as a sequence of tuples including some computation, called a 'computation-tuple sequence'. A CSS for a certain type of objects defines the set of all possible 'valid' computation-tuple sequences (object histories of the type) by specifying computation functions, semantic constraints and starting conditions.

For example, in a checking-account history, one occurrence might be, in part, the amount to be deposited or withdrawn, together with the computation of the new balance and new daily minimum balance. A CSS for objects of the type 'checking account' specifies the set of all possible 'valid' individual checking-account histories.

For our purposes, queries on object histories are just functions which map one object history to another. 'Interval' queries are those queries which, when applied to an object history, return an interval (i.e., a consecutive subsequence) of that object history. Many

Proceedings of the Tenth International Conference on Very Large Data Bases.

Singapore, August, 1984

real-life queries on object histories turn out to be interval queries. (Two examples are 'Retrieve the initial portion of Smith's checking-account history up to the first time the balance exceeds 10,000 dollars'; and 'Retrieve that portion of Smith's checking-account history dealing with June, 1983'.)

For a given CSS T, let VSEQ(T) denote the set of all possible valid object histories defined by T. Then, the main question addressed in this paper is the following:

'Given an interval query q and CSS T, when does there exist a CSS T' such that VSEQ(T')=q(VSEQ(T))?'

This question is not only of mathematical interest by itself, but also analogous to problems of 'external schema' (or 'view') [U] management which occur in traditional database situations. A CSS may be regarded as a conceptual schema which describes all the possible valid instances (here, object histories). q(VSEQ(T)) can be viewed as the set of all possible instances of an external schema defined by q and T. Then VSEQ(T')=q(VSEQ(T)) means that T' can define exactly all the possible instances of the external schema. The existence of such a T' provides a step toward the solution of the following problems:
(1) The 'implied constraint problem' [KP], which asks whether or not a constraint is true for all instances of an external schema; and
(2) The 'view update problem' [DB], which is concerned with when updates on an 'external database' (i.e., an instance in q(VSEQ(T))) can be simulated on the underlying conceptual database (i.e., some instance in VSEQ(T)).

The paper itself is divided into four sections. Section 1 reviews our model for object histories and introduces interval queries. Section 2 provides necessary and sufficient conditions on an arbitrary interval query q and CSS T such that q(VSEQ(T))=VSEQ(T') for some CSS T'. Section 3 concerns a special class of interval queries, called 'prefix' queries. (These are queries which return an 'initial' part, i.e., a prefix, of an object history.) Two different kinds of prefix queries q are presented such that for each CSS T there always exists a CSS T' satisfying VSEQ(T')=q(VSEQ(T)), with T and T' having the same computation functions. Section 4 is concerned with another important special class of interval queries, called 'suffix' queries. (These are queries which return a terminal part, i.e., a suffix, of an object history.) Two sets of conditions are presented, involving both q and T, which quarantee the existence of another CSS T' such that VSEQ(T')=q(VSEQ(T)).

1. PRELIMINARIES

In this section, we first review our model for object histories, and then discuss some preliminary concepts about queries.

Throughout, Dom_{∞} is an infinite set of elements (called domain values) and U_{∞} is an infinite set of symbols (called attributes). For each A in U_{∞}, Dom(A) (called the domain of A) is a subset of Dom_{∞} of at least two elements. All attributes considered are assumed to be elements of U_{∞}. The symbols A, B and C (possibly with subscripts) denote attributes and U (possibly subscripted) denotes a nonempty finite set of attributes. A prefix, interval and suffix of a sequence $p_1 \ldots p_m$ are subsequences of the form $p_1 \ldots p_i$, $p_i \ldots p_j$, $p_j \ldots p_m$ for some $1 \le i \le j \le m$, respectively.

Let X be a nonempty finite set of attributes and $A_1, \ldots, A_n$ some fixed listing of the distinct elements of X. Then <X> denotes the sequence $A_1 \ldots A_n$, and Dom(<X>) the cartesian product $Dom(A_1) \times \ldots \times Dom(A_n)$. Also, $<X|A_i>$ denotes the prefix $A_1 \ldots A_{i-1}$, $i \ge 2$. Let <U> be a sequence of attributes. A computation tuple over <U> is an ordered pair (<U>,u), or u when <U> is understood, where u is an element in Dom(<U>). A computation-tuple sequence over <U> is a nonempty sequence $\bar{u}$ of computation tuples over <U>. The set of all computation-tuple sequences over <U> is denoted by SEQ(<U>). Unless otherwise stated, u, v and w, possibly subscripted or primed, always represent computation tuples. Similarly, $\bar{u}$, $\bar{v}$ and $\bar{w}$ always represent computation-tuple sequences.

Given a sequence <U> of attributes, a 'computation-tuple sequence scheme' over <U> defines the set of all possible 'valid' computation-tuple sequences over <U>. It consists of
(+) a 'computation scheme', in which the attributes in U are partitioned into three types according to their roles and necessary computation functions are specified;
(++) a set of semantic constraints, called 'uniform', whose satisfaction is to hold uniformly throughout a computation-tuple sequence; and
(+++) a set of specific computation-tuple

sequences of some bounded length, called an 'initialization', with which to start a valid computation-tuple sequence.

With respect to (+), we have:
[Definition] An attribute scheme over <U> is a triple (<S>, <I>, <E>), where S, I and E are pairwise disjoint subsets of U (of state, input and evaluation attributes, resp.) with S and I nonempty and <U>=<S><I><E>.
[Here, given sequences, $<U_1>=A_1...A_{m_1}$ and $<U_2>=B_1...B_{m_2}$, $<U_1><U_2> = A_1...A_{m_1}B_1...B_{m_2}$.]
A computation scheme (abbreviated CS) over <U> is a 5-tuple $\tilde{C}=(<S>,<I>,<E>,\tilde{E},\tilde{F})$, where
(1) (<S>,<I>,<E>) is an attribute scheme over <U>;
(2) $\tilde{E}=\{e_C |$ C in E, each e_C is a partial function (called an evaluation function) from $Dom(<U>)^{\rho_C}$ X Dom(<U|C>) into Dom(C) for some non-negative integer $\rho_C\}$; and
(3) $\tilde{F}=\{f_A |$ A in S, each f_A is a partial function (called a state function) from Dom(<U>) into Dom(A)}.
The integer ρ_C is called the rank of e_C; and $\rho=\max\{\rho_C, 1 | e_C$ in $\tilde{E}\}$ is the rank of $\tilde{C}$. ■

Intuitively, the rank of a computation scheme is the minimum number of previous computation tuples on which each computation tuple computationally depends. The purpose of a computation scheme is to select those computation-tuple sequences whose values for the state and evaluation attributes are ultimately determined by the corresponding state and evaluation functions. More formally, we have:

Notation For each CS $\tilde{C}=(<S>,<I>,<E>,\tilde{E},\tilde{F})$ over <U>, let VSEQ($\tilde{C}$) be the set of all $\bar{u}=u_1...u_m (m \geq 1)$ in SEQ(<U>) satisfying the following conditions:
(1) For each $2 \leq h \leq m$ and A in S, $u_h(A)=f_A(u_{h-1})$; and
(2) For each $\rho_C<h \leq m$ and C in E, $u_h(C)=e_C(u_{h-\rho_C},...,u_{h-1},u_h[<U|C>])$. (Here, for $<U>=A_1...A_n$, <X> a subsequence of <U>, and each computation-tuple u over <U>, u[<X>] is the computation-tuple v over <X> defined by v(A)=u(A) for each A in X.)

Clearly, VSEQ($\tilde{C}$) is an interval-closed set. That is, whenever $\bar{u}$ is in VSEQ($\tilde{C}$), every interval of $\bar{u}$ is also in VSEQ($\tilde{C}$).

To formalize (++), we have:
[Definition] A constraint ω over SEQ(<U>) is a mapping over SEQ(<U>) which assigns to each $\bar{u}$ in SEQ(<U>) a value of 'true' or 'false'. If $\omega(\bar{u})$=true, then $\bar{u}$ is said to satisfy ω, denoted $\bar{u} \vdash \omega$. The set $\{\bar{u}$ in SEQ(<U>)$| \bar{u} \vdash \omega, \omega$ in $\Delta\}$ is denoted by VSEQ(Δ). A constraint ω over SEQ(<U>) is uniform if, for each $\bar{u}=u_1...u_m$ over <U>, $\bar{u} \vdash \omega$ implies $u_i...u_j \vdash \omega$ for all i and j, $1 \leq i \leq j \leq m$. ■

We shall usually define a constraint ω by just specifying VSEQ(ω). (Here, VSEQ(ω) is written instead of the more formal VSEQ($\{\omega\}$).) Since the concept of a constraint is too general, we restricted our constraints to the class of uniform constraints. These are characterized by the fact that satisfaction holds uniformly throughout a computation-tuple sequence, i.e., holds in every interval of a computation-tuple sequence. [In a checking-account history example, 'each DATE-attribute value uniquely determines its INTEREST-RATE-attribute value' is a uniform constraint.] Clearly, if Δ is a set of uniform constraints, then VSEQ(Δ) is an interval-closed set.

As for (+++), we have:
[Definition] Given a CS $\tilde{C}$ over <U> and a finite set Δ of uniform constraints over SEQ(<U>), an initialization (with respect to $\tilde{C}$ and Δ) is any prefix-closed subset $\tilde{I}$ of
$\{\bar{u}$ in VSEQ($\tilde{C}$) $\cap$ VSEQ(Δ)$| |\bar{u}| \leq \rho, \rho$ the rank of $\tilde{C}\}$,
where $|\bar{u}|$ denotes the length of $\bar{u}$. Given an initialization $\tilde{I}$, let VSEQ($\tilde{I}$) denote the set
$\tilde{I} \cup \{\bar{u}$ in SEQ(<U>)$| \bar{u}=\bar{v}\bar{w}$ for some $\bar{v}$ in $\tilde{I}$ of length $\rho\}$. ■

Intuitively, an initialization is an appropriate set of computation-tuple sequences with which to start a sequence until all state and evaluation functions can be applied. [In a checking-account history example, 'each valid checking-account history must start with DEPOSIT transaction' will be represented by an initialization.] Clearly, each VSEQ($\tilde{I}$) is prefix closed but not necessarily interval closed.

We are now ready to define a computation-tuple sequence scheme.
[Definition] A computation-tuple sequence scheme (abbreviated CSS) over <U> is a triple $T=(\tilde{C},\Delta,\tilde{I})$, where
(1) $\tilde{C}$ is a computation scheme over <U>;
(2) Δ is a finite set of uniform constraints over SEQ(<U>); and
(3) $\tilde{I}$ is an initialization with respect to $\tilde{C}$ and Δ.
For each CSS $T=(\tilde{C},\Delta,\tilde{I})$, let
VSEQ(T)=VSEQ($\tilde{C}$) $\cap$ VSEQ(Δ) $\cap$ VSEQ($\tilde{I}$).
A computation-tuple sequence is said to be valid (for T) if it is in VSEQ(T). ■

Thus, a computation-tuple sequence is valid if it (i) is 'consistent' with $\tilde{C}$, (ii) satisfies each constraint in Δ, and (iii) is either in the initialization or its prefix, of length the rank of $\tilde{C}$, is in the initialization. Since both VSEQ($\tilde{C}$) and VSEQ(Δ) are interval closed and VSEQ($\tilde{I}$) is prefix closed, VSEQ(T) is prefix closed. However, VSEQ(T) is not necessarily interval closed.

We now turn to 'queries' on object histories. For our purposes, a query is a mapping on the information set which returns either a portion of it or some 'derived' data from it. Our interest is in queries which, when applied to an object history, returns an interval of the object history. So, formally, we have:

[Definition] A <u>query</u> q over SEQ(<U>) is a (partial) mapping from SEQ(<U>) into SEQ(<U>). An <u>interval query</u> over SEQ(<U>) is a query over SEQ(<U>) such that for each $\bar{u}$ in SEQ(<U>), $q(\bar{u})$ is an interval of $\bar{u}$ if $q(\bar{u})$ exists. ■

Our formalism for interval queries is clearly of a mathematical nature. It would be interesting to implement a query language which includes many important interval queries, but this is beyond the scope of the present paper.

We conclude the section with the following examples of types of interval queries:

(Q1.1) The query Prefix_k, $k \geq 1$, which returns the first k computation tuples. That is, for each $\bar{u}=u_1...u_m$ in SEQ(<U>), $\text{Prefix}_k(\bar{u})=u_1...u_k$ if $m \geq k$, and is undefined otherwise. [One such query is 'Retrieve the initial 10 transactions in Smith's checking-account history.']

(Q1.2) The query q returning the prefix whose last computation tuple is the k-th ($k \geq 1$) computation tuple satisfying a given 'condition' in the original computation-tuple sequence, or everything if no such k-th computation tuple exists. That is, $q(\bar{u}_1\bar{u}_2)= \bar{u}_1$ ($\bar{u}_2$ is possibly empty) if the last tuple of $\bar{u}_1$ is the k-th computation tuple in $\bar{u}_1\bar{u}_2$ satisfying a given 'condition', and $q(\bar{u}_1)=\bar{u}_1$ otherwise. [One such query is 'Retrieve the part of Smith's checking-account history which starts from the first transaction and ends at the second deposit transaction whose amount is greater than 2000 dollars.']

(Q1.3) The query Chop^k, $k \geq 1$, which removes the first k computation tuples. That is, Chop^k is defined for each $\bar{u}=u_1...u_m$ in SEQ(<U>) by $\text{Chop}^k(\bar{u})=u_{k+1}...u_m$ if $m>k$, and is undefined otherwise. [One such query is 'Retrieve that part of Smith's checking-account history after the first 100 transactions.]

(Q1.4) The query returning the suffix which starts from the k-th computation tuple ($k \geq 1$) satisfying a given 'condition'. That is, $q(\bar{u}_1\bar{u}_2)=\bar{u}_2$ ($\bar{u}_1$ possibly empty) if the first computation tuple of $\bar{u}_2$ is the k-th computation tuple in $\bar{u}_1\bar{u}_2$ satisfying a given 'condition', and is undefined if no such k-th computation tuple exists. [One such query is 'Retrieve that part of Smith's checking account history which starts from the second entry for Dec. 1, 1983.']

(Q1.5) The query q returning the interval whose first computation tuple is the k_1-th ($k_1 \geq 1$) computation tuple satisfying a given 'condition' and whose last computation tuple is the k_2-th ($k_2 \geq 1$) computation tuple satisfying another (possibly the same) given 'condition'.

2. ARBITRARY INTERVAL QUERIES

In this section we present necessary and sufficient conditions on an arbitrary interval query q and CSS $T=(\tilde{C},\Delta,\tilde{I})$ such that $q(\text{VSEQ}(T))=\text{VSEQ}(T')$ for some CSS $T'=(\tilde{C}',\Delta',\tilde{I}')$ having the same attribute scheme as T. We establish two theorems. The first is for the case when $\tilde{C}'$ must be $\tilde{C}$, and the second when $\tilde{C}'$ need not be $\tilde{C}$. Both results are frequently used in the later sections.

For each subset S of SEQ(<U>), let Interval(S) denote the set $\{\bar{u} \mid \bar{u}$ an interval of some element in S$\}$. We now show our first theorem:

[Theorem 2.1] Let q be an interval query over SEQ(<U>) and $T=(\tilde{C},\Delta,\tilde{I})$ be a CSS over <U> with rank ρ of $\tilde{C}$, and let S=q(VSEQ(T)). Then there exists a CSS $T'=(\tilde{C},\Delta',\tilde{I}')$ such that VSEQ(T')=S iff S is prefix closed and
(+) $\text{Prefix}_\rho(S) \cap \text{Prefix}_\rho(\text{Interval}(S) - S) = \emptyset$.
If such a T' exists, then $\text{VSEQ}((\tilde{C},\{\omega'\},\tilde{I}'))= S$, where (i) ω' is the (uniform) constraint over SEQ(<U>) defined by $\text{VSEQ}(\omega')=\text{Interval}(S)$, and (ii) $\tilde{I}'=\{\bar{u}$ in S$\mid |\bar{u}| \leq \rho\}$. ■

[Example 2.1] Let $T=(\tilde{C},\{\omega\},\tilde{I})$ be the following CSS over <U>=ABC:
(a) $\tilde{C}$ is the CS $(A,B,C,\tilde{E},\tilde{F})$ over <U>, where (1) Dom(A)= Dom(B)= Dom(C)= N, where N is the set of nonnegative integers; (2) $\tilde{E}=\{e_C\}$, where e_C is the

function from Dom(A) x Dom(B) into Dom(C) defined by $e_C(a,b)=a+b$ for all a in Dom(A) and b in Dom(B); and (3) $\tilde{F}=\{f_A\}$, where f_A is the (partial) function from Dom(<U>) into Dom(A) defined for all c in Dom(C) by $f_A(0,0,c)=0$, $f_A(0,1,c)=1$, $f_A(1,0,c)=1$, $f_A(1,1,c)=0$, $f_A(2,3,c)=3$, $f_A(3,2,c)=1$, and undefined otherwise. [Thus, the rank ρ of $\tilde{C}$ is 1.]
(b) ω is the (uniform) constraint over SEQ(<U>) defined for each $\bar{u}=u_1...u_m$ ($m \geq 1$) by $\omega(\bar{u})$=true iff $u_i(A)=u_j(A)$ implies $u_i(B)=u_j(B)$ for all i and j, $1 \leq i<j \leq m$.
(c) $\tilde{I}=\{(000), (011), (101), (112), (235)\}$.

[In the above (c), for example, the computation-tuple (0,0,0) is written as (000) for simplicity.]
Let q be the interval query $Chop^1$. Then q(VSEQ(T)) is prefix closed. Since (011)(112) is in VSEQ(T), (112) is in $Prefix_1(q(VSEQ(T)))$. Since (235)(325)(112)(000) is in VSEQ(T), (112)(000) is in Interval(q(VSEQ(T))). And since (112)(000) is not in q(VSEQ(T)) (because of ω), (112) is in

$$Prefix_1[Interval(q(VSEQ(T))) - q(VSEQ(T))].$$

Thus, (112) is in

$$Prefix_1(q(VSEQ(T))) \cap Prefix_1[Interval(q(VSEQ(T))) - q(VSEQ(T))].$$

By Theorem 2.1, there is no CSS $T'=(\tilde{C},\Delta',\tilde{I}')$ such that VSEQ(T')=q(VSEQ(T)). ■

The statement 'q(VSEQ(T)) is prefix closed' appears in Theorem 2.1. It is readily seen that the following property, which is on q <u>alone</u>, guarantees that q(VSEQ(T)) is prefix closed for <u>every</u> CSS T.

[Definition] An interval query q over SEQ(<U>) is said to be <u>strictly downward existing</u> if for each $\bar{u}$ in SEQ(<U>) and each computation tuple u over <U>, $|q(\bar{u}u)| \geq 2$ implies that ($q(\bar{u})$ exists and)

$$q(\bar{u})=Prefix_{|q(\bar{u}u)|-1}(q(\bar{u}u)),$$

i.e., $q(\bar{u}u)=q(\bar{u})v$ for some v. ■

Examples of types of strictly-downward-existing interval queries are (Q1.2) - (Q1.4) of Section 1. Two other types are the following:

(Q2.1) The query q returning the suffix which starts from the last computation tuple satisfying a given 'condition'. That is, $q(\bar{u}_1\bar{u}_2)=\bar{u}_2$ ($\bar{u}_1$ possibly empty) if the first computation tuple of $\bar{u}_2$ is the last computation tuple in $\bar{u}_1\bar{u}_2$ satisfying a given 'condition', and undefined otherwise.

(Q2.2) The query $Chop_k$, $k \geq 1$, which returns all but the last k computation tuples. That is, for each $\bar{u}=u_1...u_m$ in SEQ(<U>), $Chop_k(\bar{u})=u_1...u_{m-k+1}$ if m>k and is undefined otherwise.

Note that the composition of strictly-downward-existing interval queries is also strictly downward existing.

Theorem 2.1 addressed the question as to when, given a CSS T and interval query q, there exists a CSS T' with the same computation scheme as T such that VSEQ(T')=q(VSEQ(T)). The next theorem considers the problem as to when there exists a CSS T', whose computation scheme is not necessarily the same as T but whose attribute scheme is the same as T, such that VSEQ(T')=q(VSEQ(T)). First though, we need a lemma which provides a basic characterization for changing computation schemes.

[Lemma 2.1] Let $T=(\tilde{C},\Delta,\tilde{I})$ be a CSS with at least one evaluation attribute and let ρ be the rank of $\tilde{C}$. Then for each integer $\rho'>\rho$, there exists a CSS $T'=(\tilde{C}',\Delta,\tilde{I}')$ with the same attribute scheme as T such that ρ' is the rank of $\tilde{C}'$ and VSEQ(T')=VSEQ(T). ■

Using Theorem 2.1 and Lemma 2.1, we now obtain:

[Theorem 2.2] Let q be an interval query over SEQ(<U>) and T a CSS with at least one evaluation attribute, and let S=q(VSEQ(T)). Then for each positive integer ρ', there exists a CS $\tilde{C}'$ of rank ρ' and a CSS $T'=(\tilde{C}',\Delta',\tilde{I}')$ such that VSEQ(T')=S iff S is prefix closed and
(+) $Pretix_{\rho'}(S) \cap Prefix_{\rho'}[Interval(S) - S] = \emptyset$.
If such a T' exists, then $S=VSEQ((\tilde{C}',\{\omega'\},\tilde{I}'))$ for some $\tilde{C}'$, with (i) ω' the (uniform) constraint over SEQ(<U>) defined by $VSEQ(\omega')=Interval(S)$, and (ii) $\tilde{I}'=\{\bar{u} \text{ in } S \mid |\bar{u}| \leq \rho'\}$. ■

[Example 2.2] Let $T=(\tilde{C},\Delta,\tilde{I})$ be the CSS given in Example 2.1. It was shown there that no CSS $T'=(\tilde{C},\Delta',\tilde{I}')$ exists such that $VSEQ(T')=Chop^1(VSEQ(T))$. Clearly, $\bar{u}$ in SEQ(<U>) is in VSEQ(T) iff $\bar{u}$ is a prefix of one of the computation-tuple sequences (for some nonnegative integer m and some positive integer n) in the following list:

$(000)^n$; $((011)(112))^n$; $(011)(101)^n$; $(101)^n$; $(112)(000)^n$; $((112)(011))^n$; $(235)(325)(101)^n$; $(235)(325)(112)(000)^n$; $(235)(325)((112)(011))^n$; $(011)(1,m,m+1)$; $(112)(0,m,m)$; $(235)(3,m,m+3)$

Thus, $\bar{u}$ in SEQ(<U>) is in $Chop^1(VSEQ(T))$ iff $\bar{u}$ is a prefix of one of the computation-tuple sequences (for some nonnegative integer m and positive integer n) in the following list:
$(000)^n$; $((112)(011))^n$; $(101)^n$; $((011)(112))^n$; $(325)(101)^n$; $(325)(112)(000)^n$; $(325)((112)(011))^n$; (1,m,m+1); (0,m,m); (3,m,m+3)

We now see if Theorem 2.2 holds for the case $\rho'=2$. Obviously,

(1) $Prefix_2(Chop^1(VSEQ(T)))$ = {(000)(000), (112)(011), (101)(101), (011)(112), (325)(101), (325)(112)} and
(2) $Prefix_2[\ Interval(\ Chop^1(\ VSEQ(T))) - Chop^1(\ VSEQ(T))] = \{(112)(000)\}$.

The intersection of (1) and (2) is empty. By Theorem 2.2, there exists a CSS $T'=(\tilde{C}',\Delta',\tilde{I}')$ with the same attribute scheme as T such that the rank of $\tilde{C}'$ is 2 and $VSEQ(T')=Chop^1(VSEQ(T))$. Furthermore, it is readily seen that one such CS $\tilde{C}'$ is $(A,B,C,\tilde{E}',\tilde{F}')$, where $\tilde{E}'=\{e'_C\}$ and $\tilde{F}'=\tilde{F}$. Here, e'_C is the function from $Dom(ABC)^2$ x Dom(AB) into Dom(C) defined by

$e'_C(a_1,b_1,c_1,a_2,b_2,c_2,a_3,b_3) = e_C(a_3,b_3)$

for each a_1, a_2, a_3 in Dom(A), b_1, b_2, b_3 in Dom(B) and c_1, c_2 in Dom(C). By Theorem 2.2, one such Δ' is $\{\omega'\}$, where ω' is the (uniform) constraint over SEQ(ABC) defined by $VSEQ(\omega')=Interval(Chop^1(VSEQ(T)))$; and one such $\tilde{I}'$ is the set

$\{\bar{u}$ in $Chop^1(VSEQ(T))\mid |\bar{u}| \le 2\}$ = {(0,m,m), (1,m,m+1), (3,m,m+3)| m ≥ 0} ∪ {(000)(000), (112)(011), (101)(101), (011)(112), (325)(101), (325)(112)}. ■

We next exhibit a CSS T for which there is no T' such that $VSEQ(T')= Chop^1(VSEQ(T))$.

[Example 2.3] Let $T=(\tilde{C},\Delta,\tilde{I})$, with $\tilde{C}=(A,B,C,\tilde{E},\tilde{F})$, be the CSS over <U>=ABC defined as follows:
(a) Dom(A)=Dom(B)=Dom(C)=N;
(b) $\tilde{E}=\{e_C\}$, where e_C is the function from Dom(A) x Dom(B) into Dom(C) defined by $e_C(a,b)=a+b$ for all a in Dom(A) and b in Dom(B);
(c) $\tilde{F}=\{f_A\}$, where f_A is the function from Dom(<U>) into Dom(A) defined by $f_A(a,b,c)=a$ for all a in Dom(A), b in Dom(B) and c in Dom(C);
(d) $\Delta=\{\omega\}$, where ω is the (uniform) constraint over <U> defined for each $\bar{u}=u_1...u_m$ in SEQ(<U>) by $\omega(\bar{u})$=true iff there exists some positive integer n such that $u_1(B)u_2(B)...u_m(B)$ is an interval of either $(0)(1)(2)(3)^n(4)$ or $(5)(2)(3)^n(6)$; and
(e) $\tilde{I}$={(000), (055)}.

Let $S=Chop^1(VSEQ(T))$. Clearly, a computation-tuple sequence is in S iff it is a prefix of either $(011)(022)(033)^n(044)$ or $(022)(033)^n(066)$ for some positive integer n. Since (022)(033)(044) is in Interval(S) - S, (022) is in $Prefix_1(Interval(S) - S)$. Since (022) is also in $Prefix_1(S)$, it follows that

(1) $Prefix_1(S) \cap Prefix_1(Interval(S) - S) \neq \emptyset$.

Now let k be any integer greater than 1. Then $(022)(033)^{k-1}(044)$ is in Interval(S)- S. Thus, $(022)(033)^{k-1}$ is in $Prefix_k(Interval(S) - S)$. Since $(022)(033)^{k-1}(066)$ is in S, $(022)(033)^{k-1}$ is also in $Prefix_k(S)$. Therefore,

(2) $Prefix_k(S) \cap Prefix_k(Interval(S) - S)\neq\emptyset$.

By (1), (2) and Theorem 2.2, there is no CSS T' with the same attribute scheme as T such that $VSEQ(T')=Chop^1(VSEQ(T))$. [Without going into details, it can readily be shown that even if the attribute scheme is changed there is no CSS T" such that $VSEQ(T")=Chop^1(VSEQ(T))$.] ■

3. PREFIX QUERIES

In this section, we consider the closure property of a special kind of interval query called 'prefix' query. In particular, we present two conditions on an arbitrary prefix query q such that for each CSS $T=(\tilde{C},\Delta,\tilde{I})$, $q(VSEQ(T))=VSEQ(T')$ for some T' of the form $(\tilde{C},\Delta',\tilde{I}')$.

A prefix query is just an interval query which returns a prefix of a given computation-tuple sequence. [Some examples of types of prefix queries are (Q1.1), (Q1.2) and (Q2.2).] Formally, we have:

[Definition] A prefix query over SEQ(<U>) is an interval query q over SEQ(<U>) such that for each $\bar{u}$ in SEQ(<U>), $q(\bar{u})$ is a prefix of $\bar{u}$ if $q(\bar{u})$ exists. ■

We now establish our first result on prefix queries.
[Theorem 3.1] Let q be a prefix query over SEQ(<U>) such that
(*) for each $\bar{u}$ of length at least 2 and each interval $\bar{v}$ of length $|\bar{u}|-1$ of $\bar{u}$, if $|q(\bar{u})| \ge 2$ then $(q(\bar{v})$ exists and) $0 \le |q(\bar{u})|-|q(\bar{v})| \le 1$.
Then for each CSS $T=(\tilde{C},\Delta,\tilde{I})$ over <U>, $VSEQ(T')=q(VSEQ(T))$, where $T'=(\tilde{C},\{\omega\},\tilde{I}')$, $VSEQ(\omega)=Interval(q(VSEQ(T)))$ and $\tilde{I}'=\{\bar{u}$ in $q(VSEQ(T))\mid |\bar{u}| \le \rho\}$, ρ the rank of $\tilde{C}$. ■

Some type of prefix query which satisfies (*) is (Q2.2). Another is:
(Q3.1) the query q which returns the first k ($k \geq 1$) computation tuples if the length of the original computation-tuple sequence is at least k, and returns everything otherwise. That is, for each $\bar{u}=u_1...u_m$ in SEQ(<U>), $q(\bar{u})= u_1...u_k$ if $m \geq k$ and $q(\bar{u})=\bar{u}$ otherwise.

A prefix query which satisfies (*) is:
(Q3.2) The query q which returns the prefix whose length is the smallest integer at least as large as half the length of the original computation-tuple sequence. That is, for each $\bar{u}=u_1...u_m$ in SEQ(<U>), $q(\bar{u})=u_1...u_{m/2}$ if m is even and $q(\bar{u})=u_1...u_{(m+1)/2}$ if m is odd.

While Theorem 3.1 is applicable to many queries, there are many for which it is not. In particular, most prefix queries of type (Q1.2) of Section 1 do not satisfy (*) of Theorem 3.1. [For example, let k=2, let v_1, v_2, w_2 be computation tuples satisfying a given condition, say depositing at least 2,000, and w_1 and w_3 computation tuples not satisfying the given condition. Then $q(v_1v_2w_1w_2w_3)=v_1v_2$ and $q(v_2w_1w_2w_3)=v_2w_1w_2$. Hence, $0 \leq |q(v_1v_2w_1w_2w_3)| - |q(v_2w_1w_2w_3)| \leq 1$ is false.] Nevertheless, as is shown by the next theorem, for that type of query and each CSS $T=(\tilde{C},\Delta,\tilde{I})$, $q(VSEQ(T))=VSEQ(T')$ for some T' of the form $(\tilde{C},\Delta',\tilde{I}')$.

In order to establish our next theorem, we need two lemmas. The first provides a condition on a prefix query q which guarantees that q can be 'equivalently replaced' by an 'all or nothing' prefix query q' having special properties.

[Lemma 3.1] Let q be a prefix query over SEQ(<U>) such that (a) q(SEQ(<U>)) is interval closed, and (b) for each $\bar{u}$ in SEQ(<U>), $q(q(\bar{u}))=q(\bar{u})$ if $q(\bar{u})$ exists. Let q' be the query over SEQ(<U>) defined by $q'(\bar{u})=\bar{u}$ if $\bar{u}$ is in q(SEQ(<U>)), and $q'(\bar{u})$ is undefined otherwise. Then q' is a prefix query such that
(i) $q'(VSEQ(T))=q(VSEQ(T))$ for each CSS T over <U>, and
(ii) if $q'(\bar{u})$ exists, then (1) $q'(\bar{u})=\bar{u}$ and (2) $q'(\bar{u}')$ exists for each interval $\bar{u}'$ of $\bar{u}$. ■

The second lemma asserts that if q satisfies (ii) of Lemma 3.1, then $q(VSEQ(T))=VSEQ(T')$ for an appropriate T'. More precisely, we have:

[Lemma 3.2] Let q be an interval query over SEQ(<U>) such that for each $\bar{u}$ in SEQ(<U>), if $q(\bar{u})$ exists then (a) $q(\bar{u})=\bar{u}$ and (b) $q(\bar{u}')$ exists for each interval $\bar{u}'$ of $\bar{u}$. Then for each CSS $T=(\tilde{C},\Delta,\tilde{I})$ over <U>, $VSEQ(T')=q(VSEQ(T))$, where $T'=(\tilde{C},\{\omega\},\tilde{I}')$, $VSEQ(\omega)=Interval(q(VSEQ(T))$ and $\tilde{I}'=\{\bar{u}$ in $q(VSEQ(T))\mid |\bar{u}| \leq \rho\}$, ρ the rank of $\tilde{C}$. ■

We are now ready for our second major result on prefix queries.

[Theorem 3.2] Let q be a prefix query over SEQ(<U>) such that (a) q(SEQ(<U>)) is interval closed, and (b) for each $\bar{u}$ in SEQ(<U>), $q(q(\bar{u}))=q(\bar{u})$ if $q(\bar{u})$ exists. Then for each CSS $T=(\tilde{C},\Delta,\tilde{I})$ over <U>, $VSEQ(T')=q(VSEQ(T))$, where $T'=(\tilde{C},\{\omega\},\tilde{I}')$, $VSEQ(\omega)=Interval(q(VSEQ(T)))$ and $\tilde{I}'=\{\bar{u}$ in $q(VSEQ(T)) \mid |\bar{u}| \leq \rho\}$, ρ the rank of $\tilde{C}$. ■

Two types of prefix queries which satisfy both (a) and (b) of Theorem 3.2 are (Q1.2) and (Q3.1).

Note that conditions (a) and (b) in Theorem 3.2 are independent of any specific CSS T. It should be also noted that:
(1) Theorem 3.2 is <u>not</u> true if the phrase 'prefix query' is replaced by 'interval query'.
(2) Condition (a) in Theorem 3.2 cannot be replaced by 'q(SEQ(<U>)) is prefix closed'.
[Illustrative examples for (1) and (2) are omitted here, but they are shown in [GT2].]

4. <u>SUFFIX QUERIES</u>

In this section, we consider the closure property of another special kind of interval query, called 'suffix' query. In Section 3, we presented conditions on an arbitrary prefix query q so that for each CSS $T(\tilde{C},\Delta,\tilde{I})$, $q(VSEQ(T))=VSEQ(T')$ for some T' of the form $(\tilde{C},\Delta',\tilde{I}')$. The situation for suffix queries is more complicated. We do not have any general theorems of the nature that if q has certain properties, then for each $T=(\tilde{C},\Delta,\tilde{I})$, $q(VSEQ(T))=VSEQ(T')$ for some T' of the form $(\tilde{C},\Delta',\tilde{I}')$ (or even for some T' which has the same attribute scheme as T). Instead our theorems here consist of conditions on q <u>and</u> T which guarantee that $q(VSEQ(T))=VSEQ(T')$ for some appropriate T'. (As will be seen, the conditions employed are on q alone, T alone and q and T in combination.) We establish two results. The first is for the case when T and T' have the same computation scheme, and the second for when T and T' only have the same attribute scheme.

A suffix query is just an interval query which returns a suffix of a given computation-tuple sequence. [Some types of

suffix queries are (Q1.3), (Q1.4) and (Q2.1).] Formally, we have:

[Definition] A suffix query over SEQ(<U>) is an interval query q over SEQ(<U>) such that for each $\bar{u}$ in SEQ(<U>), $q(\bar{u})$ is a suffix of $\bar{u}$ if $q(\bar{u})$ exists. ■

As mentioned above, the conditions needed for our major results are on q alone, T alone, and q and T in combination. For q and T in combination, we only need to assume that q(VSEQ(T)) is prefix closed as in Section 2. For T alone, we need the notions of a 'local constraint' and a 'local CSS', concepts introduced in [GT1].

[Definition] A constraint ω over SEQ(<U>) is said to be k-local ($k \geq 1$) if it has the following property: For all $\bar{u}=u_1...u_m$ ($m \geq k$), $\bar{u} \vdash \omega$ iff $u_i...u_{i+k-1} \vdash \omega$ for all $i \leq m-k+1$. A constraint is said to be local if it is k-local for some k. A CSS $T=(\tilde{C},\Delta,\tilde{I})$ is said to be local if Δ is a set of local (uniform) constraints. ■

The k-local constraints are of importance for a practical reason, namely, for ease of validity-maintenance in adding computation tuples [GT1]. Indeed, suppose ω is a k-local constraint over SEQ(<U>). Then to maintain the satisfaction of ω for adding a computation tuple u to a computation-tuple sequence $\bar{u}$ in VSEQ(ω), one need only check whether the suffix of length k of $\bar{u}u$ is in VSEQ(ω), a relatively easy task to do.

The following properties for k-local constraints were noted in [GT1]:

(1) The class of k-local constraints over SEQ(<U>), thus k-local uniform constraints over SEQ(<U>), is closed under the logical connective $\wedge$.
(2) If ω is a k-local constraint, then ω is k'-local for all k'>k.
(3) If ω is a k-local constraint and $\bar{u} \vdash \omega$ for all $\bar{u}$, $|\bar{u}|<k$, then ω is a uniform constraint.
By (2), Obviously, a CSS $T=(\tilde{C},\Delta,\tilde{I})$ is local iff for some positive integer k, Δ is a set of k-local (uniform) constraints.

The condition on q alone is the following:
[Definition] A suffix query q over SEQ(<U>) is said to be strictly upward existing if for each $\bar{u}$ in SEQ(<U>) and each computation tuple u, the existence of $q(\bar{u})$ implies that ($q(\bar{u}u)$ exists and) $q(\bar{u}u)=q(\bar{u})u$. ■

If q is strictly upward existing, then knowledge of $q(\bar{u})$ gives information on q applied to computation-tuple sequences in which $\bar{u}$ is a prefix. [Two types of strictly-upward-existing suffix queries are (Q1.3) and (Q1.4).] The class of strictly-upward-existing suffix queries is of special interest when considering the 'view update problem' mentioned in the Introduction. Suppose that q is a strictly-upward-existing suffix query and VSEQ(T')=q(VSEQ(T)) for the 'conceptual schema' T. (Thus, T' describes exactly the 'external schema' defined by q and T.) Assume that $\bar{v}$ is the current 'external database', i.e., $\bar{u}$ is in VSEQ(T) and $q(\bar{u})=\bar{v}$. Then the addition of u to $\bar{v}$ (that is, the 'addition through the view $\bar{v}$') can be simulated by the addition of u to $\bar{u}$ since $q(\bar{u}u)=\bar{v}u$ and VSEQ(T')=q(VSEQ(T)).

We are now ready for our first major result on suffix queries.

[Theorem 4.1] Let q be a strictly-upward-existing suffix query over SEQ(<U>) and $T=(\tilde{C},\Delta,\tilde{I})$ a CSS over <U> such that q(VSEQ(T)) is prefix closed. Suppose that Δ is a set of $(\rho+1)$-local constraints, where ρ is the rank of $\tilde{C}$. Then VSEQ(T')=q(VSEQ(T)), where $T'=(\tilde{C},\{\omega'\},\tilde{I}')$, ω' is the (uniform) constraint over SEQ(<U>) defined by VSEQ(ω')=Interval(q(VSEQ(T))) and $\tilde{I}'=\{\bar{u}$ in $q(VSEQ(T)) \mid |\bar{u}| \leq \rho\}$. ■

To illustrate the theorem, consider:
[Example 4.1] Let $T=(\tilde{C},\{\omega_1,\omega_2\},\tilde{I})$ be the CSS over <U>=ABC defined as follows:
(a) $\tilde{C}$ is as given in Example 2.1. [Thus, the rank ρ of $\tilde{C}$ is 1.]
(b) ω_1 is the (uniform) constraint over SEQ(<U>) defined for each $\bar{u}=u_1...u_m$ ($m \geq 1$) by $\omega_1(\bar{u})$=true iff $u_i(A)=u_{i+1}(A)$ implies $u_i(B)=u_{i+1}(B)$ for all i, $1 \leq i \leq m-1$; and ω_2 is the (uniform) constraint over SEQ(<U>) defined for each $\bar{u}=u_1...u_m$ ($m \geq 1$) by $\omega_2(\bar{u})$=true iff $u_i(B)=0$ or $u_i(B)=1$ for all i, $1 \leq i \leq m$. [Clearly, $\{\omega_1,\omega_2\}$ is a set of 2-local constraints.]
(c) $\tilde{I}=\{(011)\}$.
Then $\bar{u}$ is in VSEQ(T) iff $\bar{u}$ is a prefix of one of the following:

$(011)(101)^m$, $(011)(112)(000)^m$,
$(011)((112)(011))^m(101)^n$,
$(011)((112)(011))^m(112)(000)^n$

for some positive integers m and n. Consider the strictly-upward-existing suffix query $Chop^1$. Then $\bar{v}$ is in $Chop^1$(VSEQ(T)) iff $\bar{v}$ is a prefix of either

$(101)^m$, $(112)(000)^m$, $((112)(011))^m(101)^n$, or
$((112)(011))^m(112)(000)^n$

for some positive integers m and n. Since $Chop^1(VSEQ(T))$ is prefix closed, it follows from Theorem 4.1 that $VSEQ(T')=Chop^1(VSEQ(T))$, where $T'=(\tilde{C},\{\omega'\},\tilde{I}')$, $VSEQ(\omega')=Interval(Chop^1(VSEQ(T)))$, and $\tilde{I}'=\{(101),(112)\}$. ■

It should be noted that the assumption in Theorem 4.1 that Δ is a set of $(\rho+1)$-local constraints, ρ the rank of $\tilde{C}$, cannot be arbitrarily eliminated. [The CSS $T=(\tilde{C},\Delta,\tilde{I})$ given in Example 2.1 is not local. However, in this case, it is possible to find a local CSS $T_1=(\tilde{C},\{\omega_1\},\tilde{I})$ such that $VSEQ(T_1)=VSEQ(T)$ and ω_1 is 3-local ($VSEQ(\omega_1)=Interval(VSEQ(T))$). It is readily seen that ω_1 is <u>not</u> 2-local. Since the rank of $\tilde{C}$ is 1, Theorem 4.1 cannot be applied to T_1. Also, as shown in Example 2.1, there exists no CSS $T'=(\tilde{C},\Delta',\tilde{I}')$ such that $VSEQ(T')=Chop^1(VSEQ(T_1))=Chop^1(VSEQ(T))$. The details can be found in [GT2].]

In the spirit of Theorem 2.2, we now consider changing the computation scheme of a given local CSS but keeping the same attribute scheme. By employing Lemma 2.1 and Theorem 4.1, we get:

[Theorem 4.2] Let q be a strictly-upward-existing suffix query over SEQ(<U>) and $T=(\tilde{C},\Delta,\tilde{I})$ a local CSS over <U> such that T has at least one evaluation attribute and q(VSEQ(T)) is prefix closed. Suppose that Δ is a set of k-local constraints and ρ the rank of $\tilde{C}$. Let ω' be the (uniform) constraint over SEQ(<U>) defined by

$VSEQ(\omega')=Interval(q(VSEQ(T)))$ and

$\tilde{I}=\{\bar{u}$ in $q(VSEQ(T))$ | $|\bar{u}| \leq \max\{\rho,k-1\}\}$.

Then $q(VSEQ(T))=VSEQ(T')$ for some $T'=(\tilde{C}',\{\omega'\},\tilde{I}')$ having the same attribute scheme as T. ■

Theorem 4.2 implies the following: For each <u>local</u> CSS T with at least one evaluation attribute and each strictly-upward-existing suffix query q such that q(VSEQ(T)) is prefix closed, there exists a CSS T' with the same attribute scheme as T such that $VSEQ(T')=q(VSEQ(T))$. The following should be noted:

(1) The assumption that T is local cannot be arbitrarily removed.
(2) The condition in Theorem 4.2 that q be strictly upward existing cannot be eliminated.
[Illustrative examples are omitted, but can be found in [GT2].]

In connection with Theorems 4.1 and 4.2, the following natural question arises: What additional hypothesis will guarantee that the resulting CSS T' is local? [Strictly speaking, this question does not fall within the theme of the paper.] One suprising simple answer is to require q to be strictly downward existing, as defined in Section 2. (Since this condition implies that q(VSEQ(T)) is prefix closed, the hypothesis on q(VSEQ(T)) can be omitted.) The details are omitted, but can be found in [GT2].

ACKNOWLEDGEMENTS

The first author was supported in part by the National Science Foundation under grant MCS-792-5004. The second author was supported in part by the Sakko-kai Foundation.

REFERENCES

[A] J.F.Allen, 'Maintaining Knowledge about Temporal Intervals', CACM, Vol.26, No.11, Nov.1983, pp.832-843.

[B] J.A.Bubenko, Jr., 'The Temporal Dimension in Information Modelling', in Architecture and Models in Data Base Management Systems (G.M. Nijssen ed.), North Holland, Amsterdam, 1977.

[CW] J.Clifford and D.S.Warren, 'Formal Semantics for Time in Databases', ACM TODS, Vol.8, No.2, June 1983, pp.214-254.

[DB] U.Dayal and P.A.Bernstein, 'On the Correct Translation of Update Operations on Relational Views', ACM TODS, Vol.7, No.3, Sept.1982, pp.381-416.

[GT1] S.Ginsburg and K.Tanaka, 'Computation-tuple Sequences and Object Histories', Univ. of Southern California, Computer Science Department, Technical Report, TR-83-217, Nov. 1983.

[GT2] S.Ginsburg and K.Tanaka, 'Interval Queries on Object Histories', Univ. of Southern California, Computer Science Department, Technical Report, TR-84-302, Feb. 1984.

[K] M.R.Klopprogge, 'TERM: An Approach to Include the Time Dimension in the Entity-Relationship Model', in Proc. 2nd International Conference on Entity-Relationship Approach, Washington,D.C.

[KP] A.Klug and R.Price, 'Determining View Dependencies Using Tableaux', ACM TODS, Vol.7, No.3, Sept.1982, pp.361-380.

[U] J.D.Ullman, 'Principles of Database Systems', Second Ed., Computer Science Press, Potomac, Maryland, 1982.