Appears in: Informations Systems, Vol. 19, No. 2, pages 141-173.

Database Models for Infinite and Indefinite Temporal Information

Manolis Koubarakis*
Knowledge and Data Base Systems Laboratory
Computer Science Division
Dept. of Electrical and Computer Engineering
National Technical University of Athens
Zographou 157 73, Athens, Greece

Abstract

Representation and querying of temporal information can benefit from the integration of techniques from constraint databases, database models for indefinite information and reasoning about temporal constraints. With this perspective in mind, we present a hierarchy of temporal data models: temporal relations, generalized temporal relations and temporal tables. We study the semantics of these models and develop algebraic and calculus query languages for them. The proposed models can be useful to several novel applications include planning, scheduling, project management, medical information systems, geographical information systems and natural language processing systems.

1 Introduction

Research in temporal databases has mostly concentrated on temporal models with *definite* temporal information (e.g., "the salary of Jones was $20,000 from January 1989 to April 1989") [62, 58]. There are few exceptions to this general rule. [11] was the first paper to investigate infinite periodic data (e.g., "the commitee meets every Monday") using Datalog with function symbols. [31] also studied infinite periodic data using *temporal constraints* expressed by formulas of Presburger arithmetic. [9] presented a model which allows the representation of *qualitative temporal relations* in the database (e.g., "event A happened before event B and after January 23, 1992"). [22] studied two kinds of *indefinite temporal information*: (i) the existence of a tuple in a relation over some interval of time can be uncertain, and (ii) attribute values can be unknown during the period of existence of a tuple in a relation. [19] extended the model of [57] by allowing the beginning and ending points of the validity interval of a tuple to be indefinite. Finally, this author has proposed the model of temporal tables which exploits temporal constraints for the representation of indefinite and infinite temporal information [36].

The main argument of [36] is the following. Applications involving temporal reasoning can benefit from the development of a database model which allows the representation of *definite,*

*Current address: IC-Parc, Dept. of Computing, Imperial College, London SW7 2BZ, United Kingdom. Email: msk@doc.ic.ac.uk.

indefinite, finite and *infinite* temporal information in a single unifying framework. This can be achieved with the graceful integration of concepts and techniques from three areas:

- Database models for the representation of *indefinite information* (most prominently [26, 23]).
- The *constraint database models* of [33] and [31].
- Formalisms for reasoning with temporal constraints [1, 69, 70, 66, 65, 17, 34, 47, 41].

Independently a similar proposal has been made in [13] where the temporal database model of [36] is discussed in some length. In this paper and in [35], we extend the proposal of [36] by putting the above argument on a strong formal basis. We lay the foundations of a *hierarchy* of temporal data models: *temporal relations, generalized temporal relations* and *temporal tables*. The temporal relations model is simply a theoretical tool for providing semantics for the other two models. The model of generalized temporal relations goes a step beyond traditional temporal database models by offering the ability to represent infinite temporal information in a finite way. Finally, the model of temporal tables extends the model of generalized temporal relations by allowing indefinite temporal information.

We develop calculus and algebra query languages for every model in the proposed hierarchy. The first declarative query language we consider is *relational calculus with temporal constraints*. This language is an instance of the constraint query language framework of [33]. *Modal relational calculus with temporal constraints* extends this language for the case of indefinite temporal information. The algebraic query languages we develop have originally been discussed in [36]. The new development here is the extension of the algebra of [36] with two *modal* operators $POSS$ and $CERT$. We study the semantic properties of evaluating queries expressed in these languages. Finally by translating calculus expressions into algebraic expressions, we show that safe expressions of our calculus query languages can be evaluated bottom-up in closed form as required by [33].

Although our results are presented using database terminology, the developments of this paper will be of interest to researchers in temporal reasoning [17, 16], constraint logic programming [30] and temporal logic programming [8]. In particular, the results on semantics, declarative query languages and query evaluation can easily be rephrased for these, seemingly different, contexts.

The focus of the research in this paper is theoretical. However, the realization of our models in a future temporal database management system can be exploited by several novel applications where *temporal constraints* together with *infinite* and/or *indefinite temporal information* play an important role. These applications include planning [3], scheduling [21], medical expert systems [18], geographical information systems [5], office information systems [4] and natural language processing systems [48].

The organization of this paper is as follows. In the following section, we first discuss our assumptions about time and define our terminology. Then the basic features of temporal relations, generalized temporal relations and temporal tables are presented through a simple but detailed example. In Section 3, we introduce temporal constraints. In Section 4, the proposed hierarchy of models is defined formally. In Sections 5 and 6, we present calculus and algebra query languages for every model in the hierarchy. In Section 7 we discuss several intuitive properties of the hierarchy. The translation of safe calculus queries into algebraic queries is discussed in Section 8. Finally in Section 9 and 10, we discuss related work and future research.

2 Representing Temporal Information

In this section we present our assumptions about time and define our terminology. Then we present an example which illustrates the basic features of every model in the proposed hierarchy, and inspires the subsequent development.

2.1 Assumptions and Terminology

In this paper, time is assumed to be *linear, dense* and *unbounded.* There is a single *time line* which is isomorphic to the set of rational numbers $\mathbb{Q}$. The time entities are *points* and *intervals.* Points are *identified* with the rationals while intervals are defined as pairs of points.

The last of the above assumptions is rather unnatural in any practical setting where conventional time units are needed. It is possible to start with this abstract model of time and develop a conventional system of time units as Ladkin did in [39, 40]. Ladkin defined a time unit system (TUS) which is a natural representation of everyday time on any scale. The basic units in Ladkin's TUS are sequences of the form $[year, month, day, hour, \ldots]$ where $year, month, day, hour, \ldots$ are integers constrained appropriately. For example, the interval corresponding to the tenth day of January, 1990 is represented by $[1990, 1, 10]$. Intervals can also be built from the basic time units and the operator *convexify*. For example, the interval from March 1987 through December 1988 is $convexify([1987, 3], [1988, 12])$. Ladkin proved that his TUS is a model of the theory of time which can naturally be developed from the above assumptions.[1] Although points and intervals in a TUS are more structured and intuitive, we will identify points with rationals and intervals with pairs of rationals for the purposes of this paper. If it is desirable, our results can be reconstructed for a TUS.

We concentrate on the modeling of two kinds of temporal propositions: *events* and *facts.*[2] Facts are used to capture static aspects of the world while events are used to capture dynamic aspects. The sentences "the color of the house is red" or "the car is on service" give two examples of facts. The distinguishing property of facts is *homogeneity*: whenever a fact holds for an interval, it also holds for all of its proper subintervals.[3] In traditional historical models the homogeneity property holds for all interval relations [62]. An example of an event is "hiring a new secretary" or "taking a trip to Greece". Events can be instantaneous or have a duration. In the latter case the time over which an event occurs is *maximal*: an event cannot occur over two intervals one of which contains the other.[4]

Definite temporal information. We have *definite temporal information* when all times associated with events or facts are known to be *equal to* an *absolute time* (i.e., a point or interval on the time line). In other words, all times associated with events or facts are known to full precision in the desired level of granularity. The following are some examples of definite temporal information with respect to various time lines:

- "The car was on service throughout March, 1993" (with respect to the time line of the Gregorian calendar).
- "The car was on service for the interval $[1993, 3]$" (with respect to the time line of Ladkin's TUS).
- "The car was on service for the interval $\langle 9, 12\rangle$" (with respect to our rational time line).

Most temporal database models proposed to date can handle only definite temporal information [62]. However, if one considers advanced applications like planning [3], scheduling [21] project

[1] For more details see [40].

[2] For a richer categorization of temporal propositions see [56].

[3] The term fact has also been used by [46] with the same denotation. Facts correspond to Allen's *properties* [2] and Shoham's *liquid propositions* [56].

[4] The term event is also used by [46, 2] with the same denotation.

management [53] or medical information systems [18], one will soon realize that indefinite temporal information plays an important role as well.

Indefinite temporal information.[5] We have *indefinite temporal information* when the time associated with an event or fact is either unknown or has not been fully specified. The time associated to an event or fact can be under-specified in various ways:

- The time associated with an event or fact might be specified via a *qualitative relationship* (different than equality) to some absolute time. As an example, consider the sentence "John became manager *after* March, 1993".
- The time associated with an event or fact might be specified via a relationship to the time associated with another event or fact. In this case, the two times are related through a *qualitative* or *quantitative temporal constraint.* As an example, consider the two statements "The explosion occured *after* John left the scene" and "The explosion occured *5 to 10 minutes after* John left the scene".
- The granularity of the system time line does not match the precision to which the time associated with an event or fact is known [19]. As an example, consider storing the information "John was hired on January 25, 1993" in a system with time-stamps in the granularity of a second. Since this paper does not consider a TUS, we will not be able to give more examples of this kind of indefinite information.
- Dating techniques can be imperfect. All clocks have inherent imprecision [19].

Let us now turn to another kind of information which frequently arises in applications.

Infinite temporal information. We have *infinite temporal information* when an infinite number of times is associated with an event or fact. The following are two typical examples of infinite temporal information:

- Events that happen periodically and there is no a priori bound on the number of times they will occur. As an example, consider the sentence "The committee will meet every Monday starting today". To be able to represent this kind of information effectively, we need to devise a *finite representation* for it [11, 31]. We will not consider periodic data in this paper.
- Facts that are true over an infinite number of points or intervals. Even the ubiquitous fact "John's salary was $50,000 from January 1, 1990 to December 31, 1992" falls under this category since we have a dense model of time!

Temporal database researchers had largely ignored infinite temporal information until the recent work of [11, 12, 10, 31, 7, 6] on infinitely periodic data.

Let us now turn to an example which illustrates some of the above definitions.

2.2 Definite Temporal Information: An Example

Let us assume that we want to design a database for a transportation company. This database will be used by the scheduling personnel to keep track of orders, vehicles, loads, trips and deliveries. Two relations which can be part of this database are shown on Figures 1 and 2.

The *temporal relation* TRIP in Figure 1 is exactly like a standard relation except that the values of attribute TIME are *not* atomic (they are intervals i.e., pairs of rational numbers). TRIP

[5] A similar discussion appears in [58] and [19]. In these papers the term *indeterminate temporal information* characterizes a strict subclass of indefinite temporal information.

TRIP

VEH	FROM	TO	TIME
V3	Athens	Patra	$\langle 5, 8\rangle$
V3	Patra	Athens	$\langle 9, 12\rangle$

Figure 1: A temporal relation

ON_SERVICE

VEH	CITY	TIME	CON
V1	Athens	$i1$	$9 \leq i1_L,\ i1_R \leq 15$
V2	Athens	$i2$	$10 \leq i2_L,\ i2_R \leq 14$

Figure 2: A generalized temporal relation

represents *definite* and *finite* information about certain *events* in the world. The first tuple of TRIP states that the trip of vehicle V3 from Athens to Patra occured over the interval $\langle 5, 8\rangle$.

Let us now consider the *fact* "vehicle V1 was on service for interval $\langle 9, 15\rangle$". Notice that this implicitly says that V1 is also on service for all subintervals of interval $\langle 9, 15\rangle$. How can temporal relations represent this sort of infinite information? We will follow the approach of [31, 33] and introduce *generalized temporal relations* which are finite representations of infinite temporal relations. Generalized temporal relations, for example ON_SERVICE in Figure 2, extend temporal relations in the following way: (i) *variables* are allowed as values for temporal attributes, and (ii) every tuple is tagged with a *local temporal constraint* on its variables.

Temporal constraints are conjunctions of special kinds of formulas in the constraint language TCL which is defined in Section 3. In most cases conjuncts will be separated by commas instead of the usual "$\wedge$" operator. Temporal constraints can be used to express qualitative or quantitative properties of points and intervals. For example, in the first tuple of relation ON_SERVICE the variable $i1$ represents the interval during which vehicle V1 is in Athens for service. The local constraint of this tuple constrains interval $i1$ to be equal to or during interval $\langle 9, 15\rangle$. This containment relation is expressed by order constraints on the endpoints of the intervals $i1$ and $\langle 9, 15\rangle$. For any interval i, the left and right endpoints of i are denoted by the terms i_L and i_R respectively.

A tuple of a generalized relation can informally be understood as an *implication* where the variables are *universally quantified* and the antecedent is the local temporal constraint. Therefore a generalized relational tuple is a *finite representation* of an unrestricted (i.e., finite or infinite) set of relational tuples. For example, the first tuple of relation ON_SERVICE asserts that vehicle V1 is in Athens for service for interval $\langle 9, 15\rangle$ and all of its subintervals. This is an example of *definite* and *infinite* temporal information about some *fact* in the world. The attribute TIME of relation ON_SERVICE is semantically equivalent to *valid time* in traditional historical models [58].

Since variables in a generalized relational tuple are essentially universally quantified, they will be called *u-variables* so that they can be distinguished from e-variables, a totally different kind of variables which is introduced below.

ON_SERVICE

VEH	CITY	TIME	CON
V1	Athens	i1	$\omega 1_L \leq i1_L,\ i1_R \leq \omega 1_R$
V2	Athens	i2	$\omega 2_L \leq i2_L,\ i2_R \leq \omega 2_R$

TRIP

VEH	FROM	TO	TIME	CON
V3	Athens	Patra	$\langle 5, 8 \rangle$	true
V3	Patra	Athens	$\langle 9, \nu \rangle$	true

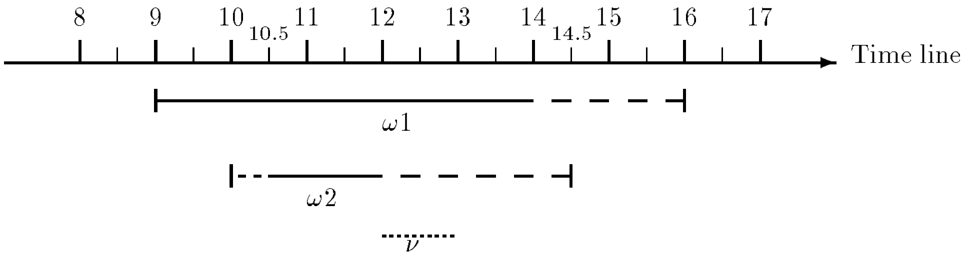

Figure 3: Temporal tables

2.3 Indefinite Temporal Information: An Example

Let us now assume that we have *indefinite* temporal information. An example of such a situation is illustrated by the *temporal tables* in Figure 3. *Temporal tables* extend generalized temporal relations by allowing the representation of indefinite information in the spirit of *conditional tables* [26, 23]. Temporal tables are exactly like generalized temporal relations but they can also have *variables* as attribute values. These variables are semantically equivalent to *marked nulls* of conditional tables or, equivalently, to *existentially quantified variables* of first-order logic. We will call them *e-variables* to distinguish them from u-variables. E-variables represent values that *exist* but are not known precisely. The *global temporal constraint* asserts all known constraints on the e-variables.

In our example, the e-variables are $\omega 1, \omega 2$ and ν.[6] The global constraint

$$G:\ \omega 1_L = 9,\ 14 \leq \omega 1_R \leq 16,\ 10 \leq \omega 2_L \leq 10.5,\ 2 \leq \omega 2_R - \omega 2_L \leq 4,\ 12 \leq \nu \leq 13$$

holds for these variables. The variables $\omega 1$ and $\omega 2$ denote time intervals whose duration is not known precisely. The variable ν denotes a time point whose position in the time line is not known exactly. The global constraint G states that (i) interval $\omega 1$ starts at time 9 and ends at some time between 14 and 16, (ii) interval $\omega 2$ starts at some time between 10 and 10.5 and its duration is 2 to 4 time units, and (iii) point ν is between 12 and 13. This information is also presented graphically in Figure 3. Intervals are represented by line segments. Solid line segments depict the extent of an interval about which we have definite information. Dashed line segments depict the

[6] A note on typography: Interval u-variables will be denoted by $i, i1, i2, \ldots$. Point u-variables will be denoted by $p, p1, p2, \ldots$. Interval e-variables will be denoted by $\omega, \omega 1, \omega 2, \ldots$. Point e-variables will be denoted by $\nu, \nu 1, \nu 2, \ldots$.

extent of an interval about which we have indefinite information. Similar conventions are followed for time points.

Temporal tables are syntactic devices for the representation of indefinite (finite or infinite) temporal data. Each table is semantically equivalent to a set of possible worlds which capture all the possible ways the domain of application could have been according to our indefinite information. Each possible world can be represented by a generalized temporal relation. One can find all the possible worlds represented by a table as follows:

- Find an assignment to the e-variables which satisfies the global constraint.
- Substitute these values for the e-variables in the table.

For example, the relation ON_SERVICE in Figure 2 represents one of the possible worlds corresponding to table ON_SERVICE.

Let us now consider the query "Find all vehicles that are on service throughout the first hour of the afternoon". Using the table calculus of Section 5.2, this query can be phrased as follows:

$$\{ANS1(VEH), x/\mathcal{D} :\ (\exists i/\mathcal{I})(\exists c/\mathcal{D})(ON\_SERVICE(x, c, i)\ \wedge\ i_L \leq 12\ \wedge\ i_R \geq 13)\}.$$

Each table calculus expression has two parts. The first part gives the name and the scheme of a table where the answer to the query will be stored. The second part is a declarative specification of what the user wants in a first-order temporal logic defined in Section 5. In our case, we want all values x such that there exist values i and c such that x is on service in city c throughout interval i. Interval i should also *contain* interval $\langle 12, 13\rangle$ (the first hour of the afternoon). The sort of x and c must be $\mathcal{D}$ i.e., they must be data values. The sort of i must be $\mathcal{I}$ i.e., it must be an interval value.

What is the answer to this query ? If we examine the database in Figure 3, we can see that vehicle V1 satisfies the qualification of the query. The situation is more involved with V2. Observe that it is possible that $\omega 2$ extends until time 13. In this case V2 is in the answer as well. But what if $\omega 2$ does not extend until time 13?

The above situation is quite typical for a query posed to a database with indefinite information. There are objects that qualify as answers to the posed query in all possible worlds represented by the database. There are also objects that qualify in one possible world but do not qualify in another. Therefore the answer to a query must be *conditional* [26]. For every object that qualifies as an answer *only in a subset* of the possible worlds represented by the database, the query answering procedure should also provide a *condition* characterizing this possible world subset.

In our example, vehicle V1 qualifies in all possible worlds represented by the database. Vehicle V2 qualifies only in the possible worlds where interval $\omega 2$ extends until time 13 or later. Therefore the answer to this query should be the following table ANS1.

ANS1

VEH	CON
V1	*true*
V2	$\omega 2_R \geq 13$

The global constraint associated with ANS1 is the initial constraint G. Queries do not alter the global constraint associated with the input database.

The utility of the local temporal constraints must be obvious now. If a local temporal constraint contains u-variables, it provides a finite representation of a possibly infinite set of tuples (e.g., witness table ON_SERVICE). If it contains only e-variables, it is a condition under which its associated tuple belongs in the table. In ANS1 the condition for tuple (V1) is *true*. The condition for tuple (V2) is $\omega 2_R \geq 13$. The set of possible worlds represented by ANS1 is given by the set of relations $\left\{ \begin{array}{|c|} \text{V1} \end{array}, \begin{array}{|c|} \text{V1} \\ \text{V2} \end{array} \right\}$. Each one of these relations represents one set of vehicles that are possibly on service throughout the first hour of the afternoon.

In some cases the user might know that the information in the database is indefinite. Thus he (or she) might wish to find all tuples that *possibly* or *certainly* satisfy some qualification. For example, let us consider the query "Find all vehicles that are possibly on service throughout the first hour of the afternoon". In table calculus, this query can be phrased as follows:

$$\{ANS2(VEH), x/\mathcal{D}: \ \diamond((\exists i/\mathcal{I})(\exists c/\mathcal{D})(ON\_SERVICE(x,c,i) \ \wedge \ i_L \leq 12 \ \wedge \ i_R \geq 13))\}.$$

This query uses the *modal possibility* operator $\diamond$. The answer to this query is the following table ANS2.

ANS2

VEH	CON
V1	*true*
V2	*true*

Now the constraint for V2 is true indicating that *unconditionally* V2 is possibly on service throughout the first hour of the afternoon.

The tables returned as answers to queries involving $\diamond$ are called *u-tables* because they do not contain e-variables. A u-table represents a *single* possible world. A representation of this possible world can be obtained if we discard the global constraint of the u-table to obtain a generalized relation.

We can also pose queries asking for all tuples that *certainly* satisfy a given query expression. In this case, we have to use the modal operator $\square$ for *necessity*. Let us consider the query "Find all times during which vehicles are certainly on service; output vehicle identification and time". This query can be phrased as follows:

$$\{ANS3(VEH, TIME), x/\mathcal{D}, i/\mathcal{I}: \ \square((\exists c/\mathcal{D})ON\_SERVICE(x,c,i))\}.$$

The answer to this query is the following table ANS3:

ANS3

VEH	TIME	CON
V1	$i1$	$i1_L \leq 10, \ i1_R \leq 14$
V2	$i2$	$i2_L \leq 10.5, \ i2_R \leq 12$

Similarly with ANS2 this table does not contain e-variables. Thus it represents a single possible world. When table ANS3 is viewed as a generalized relation, it is a finite representation of the infinite information in this possible world.

3 Temporal Constraints

In this section we define temporal constraints formally. As we have already said, we assume that time is linear, dense and unbounded. The time entities will be points and intervals. Points are identified with the rational numbers while intervals are defined as pairs of points.

The language TCL will allow us to make statements about time. TCL is a many-sorted first order calculus with equality [20]. It has two sorts: $\mathcal{P}$ (for points or rationals) and $\mathcal{I}$ (for intervals). The logical symbols of TCL include parentheses, a countably infinite set of variables V_s for each sort s, an equality symbol $=_s$ for each sort s and the standard sentential connectives.[7] The non-logical symbols of TCL include a countably infinite set of function symbols of arity 0 and sort $\mathcal{P}$ (the point constants i.e., the rational numerals), function symbols L and R of sort $(\mathcal{I}, \mathcal{P})$, function symbol $-$ of sort $(\mathcal{P}, \mathcal{P}, \mathcal{P})$ and predicate symbol $<$ of arity 2 with sort $(\mathcal{P}, \mathcal{P})$.

The symbols of TCL are interpreted with respect to the many-sorted structure $\mathcal{T}$ which encompasses our assumptions for time. $\mathcal{T}$ assigns to the sort $\mathcal{P}$ the set of rationals numbers $\mathbb{Q}$ and to the sort $\mathcal{I}$ the set $Int(\mathbb{Q}) = \{\langle r_1, r_2 \rangle : r_1, r_2 \in \mathbb{Q} \text{ and } r_1 <_{\mathbb{Q}} r_2\}$.[8] The set associated with each sort s is called its domain and is denoted by $dom(s)$. To each function symbol of arity 0 and sort $\mathcal{P}$, $\mathcal{T}$ assigns an element of $\mathbb{Q}$ (a rational number). To function symbols L and R, $\mathcal{T}$ assigns the functions $l, r : Int(\mathbb{Q}) \rightarrow \mathbb{Q}$ such that $l((r_1, r_2)) = r_1$ and $r((r_1, r_2)) = r_2$. These functions map each interval to its left and right endpoint respectively. To function symbol $-$, $\mathcal{T}$ assigns the function $-_{\mathbb{Q}}$ which is the subtraction operation for rationals. To predicate symbol $<$, $\mathcal{T}$ assigns the relation $<_{\mathbb{Q}}$ over the rationals.

We assume that the reader is familiar with the concepts of *variable assignment, satisfaction* and *truth* in many-sorted structures [20].

The set of *simple terms* of TCL is defined by the following rules:

- A variable of sort $\mathcal{P}$ is a simple term.
- If i is a variable of sort $\mathcal{I}$ then i_L and i_R are simple terms.

Simple terms are used in definining temporal constraints.

Definition 3.1 A *temporal constraint* is a formula of TCL of the following form:

- an inequality $t_1 - t_2 \leq r$ or $t_1 - t_2 < r$ where t_1, t_2 are simple terms and r is a constant of sort $\mathcal{P}$ (t_1 or t_2 can be omitted).
- *true* or *false* with the obvious semantics.

Definition 3.2 A *point constraint* is a constraint which involves only terms of sort $\mathcal{P}$.

Example 3.1 The following are examples of temporal constraints.

$$i_R - i_L \leq 6.5, \; i_R > p_1, \; p_1 - p_2 \geq 7$$

The constraint $p_1 - p_2 \geq 7$ is a point constraint.

Definition 3.3 Let C be a set of temporal constraints in variables $x_1, \ldots, x_n$ of sorts $s_1, \ldots, s_n$. The *solution set* of C is defined as follows:

$$Sol(C) = \{(a_1, \ldots, a_n) : a_i \in dom(s_i) \text{ and for each } c \in C \;\; \mathcal{T} \models c[x_1 \leftarrow a_1, \ldots, x_n \leftarrow a_n]\}.$$

Each member of $Sol(C)$ is called a *solution* of C. A set of temporal constraints is called *consistent* or *satisfiable* iff $Sol(C) \neq \emptyset$. Two sets of constraints are called *equivalent* iff they have the same solution set.

[7] For simplicity, we will use $=$ whenever the sort of the involved terms is understood.

[8] The symbol $<_{\mathbb{Q}}$ denotes the relation "less than" over the rationals.

A set of constraints is the algebraic counterpart of the logical conjunction of its members. Thus, in the following sections, we will frequently mix the terms "set of constraints" and "conjunction of constraints".

The notion of projection of a solution set will be important in the rest of this section.

Definition 3.4 Let C be a set of constraints in variables $X = \{x_1, \ldots, x_n\}$. If $V = \{x_{j_1}, \ldots, x_{j_m}\}$ is a subset of X then the *projection* of $Sol(C)$ on V, denoted by $Proj_V(Sol(C))$, is the set

$$\{ \ (a_{j_1}, \ldots, a_{j_m}) : \ (a_1, \ldots, a_n) \in Sol(C) \text{ and } \{j_1, \ldots, j_m\} \subseteq \{1, \ldots, n\} \ \}.$$

Temporal constraints are powerful enough to express a large subset of the interval-to-interval relations in Allen's algebra [2] and point-to-interval relations in Meiri's algebra [47]. Our desire to keep the constraint language as simple as possible led us to avoid introducing special predicate symbols for the relations of these algebras. We do not consider more powerful interval constraints due to the intractability of determining consistency of such constraints [70, 47] (e.g., the constraint "interval i is before or after interval j" can not be expressed in TCL.) Finally observe that although temporal constraints can involve interval variables, they are simply point or endpoint constraints. This immediately raises the question: Is it important to use interval variables ? The answer to this question is in the affirmative. However, a thorough discussion of the issue is postponed until Section 6.4.

3.1 From Temporal Constraints To Point Constraints

We will now consider translating temporal constraints into point constraints and vice versa. These translations will be useful in the forthcoming section where we discuss eliminating variables from temporal constraints. The translation of formulas involving intervals to formulas involving points has been a standard technique in temporal reasoning literature [69, 64, 41]. Our approach is closely related to [41] who considered translating formulas of the first-order theory of the binary qualitative interval-to-interval relations defined by Allen [2]) to formulas in the first-order theory of the structure $(\mathbb{Q}, <)$.

We will first define the translation function *pointise* which maps a set of temporal constraints into an "equivalent" set of point constraints. Let us assume that C is a set of temporal constraints in variables $(p_1, \ldots, p_n, i_1, \ldots, i_m)$ where $p_1, \ldots, p_n$ are of sort $\mathcal{P}$ and $i_1, \ldots, i_m$ are of sort $\mathcal{I}$. Let $B_{i_1}, \ldots, B_{i_m}$ and $E_{i_1}, \ldots, E_{i_m}$ be brand new variables of sort $\mathcal{P}$. The set of point constraints $pointise(C)$ has variables $(p_1, \ldots, p_n, B_{i_1}, \ldots, B_{i_m}, E_{i_1}, \ldots, E_{i_m})$ and is defined as follows:

$$pointise(C) = \{pt(c) : \ c \in C\} \cup \{B_{i_1} < E_{i_1}, \ \ldots, B_{i_m} < E_{i_m}\}$$

where $pt(c)$ is the same as c except that terms i_{kL} and i_{kR} have been substituted by variables B_{i_k} and E_{i_k} respectively (for all k, $1 \leq k \leq m$).

The following proposition relates constraint sets C and $pointise(C)$. Its proof is straightforward.

Proposition 3.1 *Let C be a set of temporal constraints in variables $(p_1, \ldots, p_n, i_1, \ldots, i_m)$ where $p_1, \ldots, p_n$ are of sort $\mathcal{P}$ and $i_1, \ldots, i_m$ are of sort $\mathcal{I}$. If $p_1^0, \ldots, p_n^0 \in \mathbb{Q}$ and $i_1^0, \ldots, i_m^0 \in Int(\mathbb{Q})$ then $(p_1^0, \ldots, p_n^0, i_1^0, \ldots, i_m^0)$ is a solution of C iff $(p_1^0, \ldots, p_n^0, l(i_1^0), \ldots, l(i_m^0), r(i_1^0), \ldots, r(i_m^0))$ is a solution of $pointise(C)$ where l and r are functions defined by structure $\mathcal{T}$.*

Associated with *pointise* is the inverse translation function $pointise^{-1}$. Let C be a set of point constraints in variables $(p_1, \ldots, p_n, B_{i_1}, \ldots, B_{i_m}, E_{i_1}, \ldots, E_{i_m})$. Then $pointise^{-1}(C)$ is a set of temporal constraints in variables $(p_1, \ldots, p_n, i_1, \ldots, i_m)$ where $i_1, \ldots, i_m$ are variables of sort $\mathcal{I}$ and is defined as follows:

$$pointise^{-1}(C) = \{temp(c) : \; c \in C\}$$

where $temp(c)$ is the same as c except that variables B_{i_k} and E_{i_k} have been substituted by terms i_{kL} and i_{kR} respectively.

The following propostion relates constraint sets C and $pointise^{-1}(C)$. Its proof is straightforward.

Proposition 3.2 *Let C be a set of point constraints in variables $(p_1, \ldots, p_n, B_{i_1}, \ldots, B_{i_m}, E_{i_1}, \ldots, E_{i_m})$. If $p_1^0, \ldots, p_n^0, \beta_1, \ldots, \beta_m, \epsilon_1, \ldots, \epsilon_m \in \mathbb{Q}$ then $(p_1^0, \ldots, p_n^0, \beta_1, \ldots, \beta_m, \epsilon_1, \ldots, \epsilon_m)$ is a solution of C iff $(p_1^0, \ldots, p_n^0, \langle \beta_1, \epsilon_1 \rangle, \ldots, \langle \beta_m, \epsilon_m \rangle)$ is a solution of $pointise^{-1}(C)$.*

We will now turn our attention to the problem of variable elimination.

3.2 Variable Elimination

The operation of variable elimination is the algebraic analogue of the projection operation of geometry. Let C be a set of constraints in variables $x_1, \ldots, x_n$. Eliminating variables $x_1, \ldots, x_k, \; k < n$ from C is equivalent to projecting $Sol(C)$ on the solution subspace defined by variables $x_{k+1}, \ldots, x_n$. Variable elimination will play an important part in some of the algebraic operations of the temporal data models discussed in Section 4.

For point constraints variable elimination can be performed using Fourier's algorithm which was originally devised for linear inequality constraints over the real numbers [55]. Fourier's algorithm can be summarized as follows. Any weak linear inequality involving a variable x can be written in the form $x \leq r_u$ or $x \geq r_l$ i.e., it gives an upper or a lower bound on x. Thus if we are given two linear inequalities, one of the form $x \leq r_u$ and the other of the form $x \geq r_l$, we can eliminate x and obtain the inequality $r_l \leq r_u$. Obviously, $r_l \leq r_u$ is a logical consequence of the given inequalities. In addition, any solution of $r_l \leq r_u$ can be extended to a solution of the given inequalities (simply by choosing for x any value between the values of r_l and r_u). Following this observation, Fourier's elimination algorithm forms all pairs $x \leq r_u$ and $x \geq r_l$, eliminates x and returns the resulting constraints. The generalization of this algorithm to strict linear inequalities is obvious.

Example 3.2 Eliminating variable x_1 from the set of inequalities

$$\{x_3 \leq x_1, \; x_5 \leq x_1, \; x_1 - x_2 \leq 2, x_4 \leq x_5\}$$

gives

$$\{x_3 - x_2 \leq 2, \; x_5 - x_2 \leq 2, \; x_4 \leq x_5\}.$$

For the case of temporal constraints variable elimination can be performed by translating into point constraints, eliminating variables using Fourier's algorithm and translating the resulting constraint set back into temporal constraints [35].

4 A Hierarchy of Temporal Data Models

The examples of Sections 2.2 and 2.2 must have given the reader the flavour of the models in the proposed hierarchy. Now we turn to formal definitions.

Let us assume the existence of three countably infinite sets: the set of *interval attributes* U_I, the set of *point attributes* U_P and the set of *data attributes* U_D. Elements of $U_I \cup U_P$ are also called *temporal attributes*. The set of attributes U is defined as $U = U_I \cup U_P \cup U_D$. If $A \in U$ then $sort(A)$ denotes the sort of A. If $A \in U_D$ then $sort(A) = \mathcal{D}$. If $A \in U_P$ then $sort(A) = \mathcal{P}$. Finally if $A \in U_I$ then $sort(A) = \mathcal{I}$. For each $A \in U$, we define a countably infinite set $dom(A)$, the *domain* of A. If $A \in U_D$ then $dom(A) = Dom$ (the *data domain*). If $A \in U_I$ then $dom(A) = Int(\mathbb{Q})$. Finally, if $A \in U_P$ then $dom(A) = \mathbb{Q}$. For a set of attributes $S = \{A_1, ..., A_n\}$, we define $dom(S) = dom(A_1) \cup \cdots \cup dom(A_n)$.

Definition 4.1 A *scheme* is a subset of U. If S is a scheme then S_d and S_t denote the set of data and temporal attributes of S respectively.

Let us now define temporal relations.

Definition 4.2 A *temporal relational tuple* (or simply *relational tuple*) t over scheme S is a mapping from S to $Dom \cup Int(\mathbb{Q}) \cup \mathbb{Q}$ such that $t(A) \in dom(A)$ for each $A \in S$.

For each relational tuple t over S $t(S_d)$ is called its *data part* while $t(S_t)$ is called its *temporal part*.

Definition 4.3 A *temporal relation* (or simply *relation*) R over scheme S is an *unrestricted* (i.e., finite or infinite) set of relational tuples over S.

The scheme of a relation R will be denoted by $sch(R)$. The set of relations over scheme S will be denoted by $REL(S)$.

Example 4.1 The relation TRIP in Figure 1 is an example of a temporal relation.

Let us now assume the existence of four disjoint countably infinite sets of *variables* whose intersection with the domains of attributes is empty. These are the sets of *point u-variables* V_u^p, *interval u-variables* V_u^i, *point e-variables* V_e^p and *interval e-variables* V_e^i. We assume that V_e^p and V_u^p are subsets of $V_\mathcal{P}$ and V_e^i and V_u^i are subsets of $V_\mathcal{I}$ (the sets $V_\mathcal{P}$ and $V_\mathcal{I}$ have been defined in Section 3). Let us define $V_e = V_e^p \cup V_e^i$ and $V_u = V_u^p \cup V_u^i$.

Let $TCONS(V)$ denote the set of all conjunctions of temporal constraints in the language TCL with variables from the set V.

We are now ready to define generalized temporal relations.[9]

Definition 4.4 A *generalized temporal relational tuple* (or simply *generalized relational tuple*) t over scheme S is a mapping from $S \cup \{CON\}$ to $Dom \cup V_u \cup TCONS(V_u)$ such that (i) $t(A) \in Dom$ for each $A \in S_d$, (ii) $t(A) \in V_u^p$ for each $A \in S_t \cap U_P$, (iii) $t(A) \in V_u^i$ for each $A \in S_t \cap U_I$, (iv) $t(A_1) \neq t(A_2)$ for all $A_1, A_2 \in S_t$ such that $A_1 \neq A_2$ and (v) $t(CON) \in TCONS(\{t(A) : A \in S_t\})$.

If t is a generalized relational tuple over S then $t(CON)$ is called the *local constraint* of the tuple t while $t(S)$ is called the *proper part* of t. The proper part of a tuple t is further divided into the *data part* $t(S_d)$ and the *temporal part* $t(S_t)$.

Definition 4.5 A *generalized temporal relation* (or simply *generalized relation*) R over S is a *finite* set of generalized relational tuples over S.

The scheme of a generalized relation R will be denoted by $sch(R)$. The set of all generalized relations over scheme S will be denoted by $GREL(S)$.

[9] In the following definition, condition (iv) is not necessary. It is only introduced for making some of the machinery in the rest of the paper simpler.

Example 4.2 The relation ON_SERVICE shown in Figure 2 is an example of a generalized relation.

Let us now define temporal tables.

Definition 4.6 A *temporal table tuple* (or simply *table tuple*) t over scheme S is a mapping from $S \cup \{CON\}$ to $Dom \cup V_u \cup TCONS(V_u \cup V_e)$ such that (i) $t(A) \in Dom$ for each $A \in S_d$, (ii) $t(A) \in V_u^p$ for each $A \in S_t \cap U_P$, (iii) $t(A) \in V_u^i$ for each $A \in S_t \cap U_I$, (iv) $t(A_1) \neq t(A_2)$ for all $A_1, A_2 \in S_t$ such that $A_1 \neq A_2$ and (v) $t(CON) \in TCONS(V_e \cup \{t(A) : A \in S_t\})$.

For table tuples, the notions of local constraint, proper part, data part and temporal part are defined as for generalized relational tuples.

Definition 4.7 A *temporal table* (or simply *table*) over scheme S is a *finite* set of table tuples over S. Each table T is associated with a constraint $G(T)$ from $TCONS(V_e)$. $G(T)$ is called the *global constraint* of T.

The scheme of a table T will be denoted by $sch(T)$. The set of all tables over scheme S is denoted by $TAB(S)$.

Example 4.3 ON_SERVICE and TRIP, shown in Figure 3, are examples of tables.

Definition 4.6 does not allow constants and e-variables to be values of temporal attributes of table tuples (as in the table TRIP in Figure 3). This condition is not necessary but makes some of the definitions in the rest of the paper simpler. Any table which does not obey this condition can easily be transform into an equivalent one, which respects the condition, by adding appropriate constraints to its tuples.

Let us now define a special class of tuples and tables.

Definition 4.8 A *u-tuple* t is a tuple whose local constraint $t(CON)$ does not contain variables from V_e. A *u-table* is a table containing only u-tuples.

Notice that u-tables are *not* generalized relations: they do not contain e-variables but they do have an associated global constraint.

Example 4.4 The table ANS3 of Section 2.3 is an example of a u-table.

Remark 4.1 Schemes will usually be named. A *name* is just a symbol (possibly subscripted). If $\{C_1, \ldots, C_n\}$ is a scheme named R, we will usually write this scheme as $R(C_1, \ldots, C_n)$. Relation schemes will usually be named R. Table schemes will usually be named T. Thus we overload symbols like R and T by using them to denote scheme names but also instances.

Let us now define database schemes and databases.

Definition 4.9 A *database scheme* $\widetilde{S}$ is a sequence $(S_1, \ldots, S_n)$ where each S_i is a scheme.

Definition 4.10 A *temporal table database* (or simply *table database*) $\widetilde{T}$ over scheme $(S_1, \ldots, S_n)$ is a sequence $(T_1, \ldots, T_n)$, where each T_i is a table on scheme S_i. Each table database $\widetilde{T}$ is associated with a constraint from $TCONS(V_e)$. This constraint, denoted by $G(\widetilde{T})$, is called the *global constraint* of $\widetilde{T}$. By definition $G(T_i) = G(\widetilde{T})$, $1 \leq i \leq n$. In other words, the global constraint is common to all tables in a database $\widetilde{T}$.

The *scheme* of a table database $\widetilde{T}$ will be denoted by $sch(\widetilde{T})$. The set of all databases over scheme $\widetilde{S}$ is denoted by $TAB(\widetilde{S})$. The set of all table databases (over any scheme) is denoted by TAB.

Similarly, we can define *temporal relational databases* (or simply *relational databases*) to be sequences of temporal relations, and *generalized temporal relational databases* (or simply *generalized relational databases*) to be sequences of generalized temporal relations. The set of all relational databases over scheme $\widetilde{S}$ is denoted by $REL(\widetilde{S})$. The set of all relational databases (over any scheme) is denoted by REL. The set of all generalized relational databases over scheme $\widetilde{S}$ is denoted by $GREL(\widetilde{S})$. The set of all generalized relational databases (over any scheme) is denoted by $GREL$.

4.1 Closed-World Semantics

Let us now give semantics to the concepts defined above. We will first define the concepts of e-valuation and u-valuation.

Definition 4.11 An *e-valuation* (valuation of e-variables) is a function $v : V_e \rightarrow \mathbb{Q} \cup Int(\mathbb{Q})$ such that $v(a) \in \mathbb{Q}$ if $a \in V_e^p$ and $v(a) \in Int(\mathbb{Q})$ if $a \in V_e^i$. Similarly, we define a u-valuation (valuation of u-variables) as a function $v : V_u \rightarrow \mathbb{Q} \cup Int(\mathbb{Q})$ such that $v(a) \in \mathbb{Q}$ if $a \in V_u^p$ and $v(a) \in Int(\mathbb{Q})$ if $a \in V_u^i$.

We will denote by Val^e and Val^u the set of e-valuations and u-valuations respectively. The *concatenation* of a u-valuation u and a e-valuation v is the function $u;v : V_e \cup V_u \rightarrow \mathbb{Q} \cup Int(\mathbb{Q})$ such that $(u;v)(a) = u(a)$ if $a \in V_u$ and $(u;v)(a) = v(a)$ if $a \in V_e$.

The result of applying an e-valuation v to a table T over S will be denoted by $v(T)$. $v(T)$ is a generalized relation over S obtained from T by substituting each e-variable ω of T by the constant symbol whose denotation in structure $\mathcal{T}$ is $v(\omega)$.

The result of applying a u-valuation to the proper part of a tuple can be defined as follows. If t is a tuple on scheme S and u is a u-valuation then $u(t)$ is a relational tuple over S. In addition, for each $A \in S$, $u(t)(A) = u(t(A))$ if $t(A) \in V_u$ and $u(t)(A) = t(A)$ otherwise.

We will first define the function *rep* which gives the set of generalized relations represented by a table T:

$$rep(T) = \{R \in GREL(S) : \text{ there exists valuation } v \in Val^e \text{ s.t. } \mathcal{T} \models G(T)[v] \text{ and } R = v(T)\}$$

Example 4.5 Let us consider the table ON_SERVICE in Figure 3. One of the generalized relations in $rep(ON\_SERVICE)$ is the relation with the same name in Figure 2.

The function *points* takes as argument a generalized relation R over S and returns the relation over S which is finitely represented by R. This function is defined as follows:

$$points(R) = \{u(t) : \ t \in R, u \in Val^u \text{ and } \mathcal{T} \models t(CON)[u]\}.$$

Example 4.6 Let us consider the generalized relation ON_SERVICE in Figure 2. The temporal relation $points(ON\_SERVICE)$ contains all the tuples (V1, Athens, i_1) such that $9 \leq i1_L \wedge i1_R \leq 15$ is true, and all the tuples (V2, Athens, i2) such that $10 \leq i2_L \wedge i2_R \leq 14$ is true.

We define the semantics of a table T over scheme S to be the following set of relations:

$$sem(T) = \{R \in REL(S) : \text{ there exists } R' \in rep(T) \text{ and } R = points(R')\}.$$

The above definition implies that tables are interpreted in a *closed-world* fashion with respect to their data part. In other words, tables are assumed to represent all events or facts relevant to an application domain. However, the exact times associated with these events or facts may not be known precisely.

Example 4.7 In ANS1 is the table from Section 2.3 then

$$sem(\text{ANS1}) = \left\{ \begin{array}{|l|}\hline \text{VEH} \\ \hline \text{V1} \\ \\ \hline \end{array} \; , \; \begin{array}{|l|}\hline \text{VEH} \\ \hline \text{V1} \\ \hline \text{V2} \\ \hline \end{array} \right\}.$$

Note that $sem(T) = \emptyset$ iff $G(T)$ is inconsistent. Let us then assume that $G(T)$ is consistent and for each valuation $v \in Val^e$ such that $\mathcal{T} \models G(T)[v]$, there is no tuple $t \in T$ and valuation $u \in Val^u$ such that $\mathcal{T} \models t(CON)[u; v]$. Then $sem(T) = \{\emptyset\}$.

The functions $rep, points$ and sem can be extended to databases in the following way. If $\widetilde{T} = (T_1, \ldots, T_n)$ is a table database over scheme $\widetilde{S} = (S_1, \ldots, S_n)$ then

$$rep(\widetilde{T}) = \{\widetilde{R} \in GREL(\widetilde{S}) : \text{ there exists valuation } v \in Val^e \text{ s.t. } \mathcal{T} \models G(\widetilde{T})[v] \text{ and } \widetilde{R} = v(\widetilde{T})\}.$$

If $\widetilde{R} = (R_1, \ldots, R_n)$ is a generalized relational database over $\widetilde{S}$ then $points(\widetilde{R})$ is a relational database over $\widetilde{S}$ which is defined as follows:

$$points(\widetilde{R}) = (points(R_1), \ldots, points(R_n)).$$

The semantics of a table database $\widetilde{T}$ over scheme $\widetilde{S}$ is the following set of relational databases:

$$sem(T) = \{\widetilde{R} \in REL(\widetilde{S}) : \text{ there exists } \widetilde{R}' \in rep(\widetilde{T}) \text{ and } \widetilde{R} = points(\widetilde{R}')\}.$$

Note also that $sem(\widetilde{T}) = \emptyset$ iff $G(\widetilde{T})$ is inconsistent. Let us then assume that $G(\widetilde{T})$ is consistent and for each valuation $v \in Val^e$ such that $\mathcal{T} \models G(\widetilde{T})[v]$ there is no tuple $T \in \widetilde{T}$, tuple $t \in T$ and valuation $u \in Val^u$ such that $\mathcal{T} \models t(CON)[u; v]$. Then $sem(\widetilde{T}) = (\emptyset, \ldots, \emptyset)$.

Remark. Notice that constants and e-variables are not allowed as values of temporal attributes of tuples (as in the table TRIP in Figure 3). In addition, the u-variables of a tuple are assumed to be different from each other. These are not real restrictions of our model; they only make some of the definitions simpler. Whenever a table which does not obey these rules is encountered, we can transform it to an equivalent table which respects these rules by adding appropriate constraints to its tuples.

5 Declarative Query Languages

In this section, we present a declarative query language for each of the database models discussed in Section 4. At first, we present a first-order language which will be helpful for defining the query languages.

Given a database scheme $\widetilde{S}$, we define a first-order temporal language over predicate vocabulary $\widetilde{S}$. This language will be denoted by $FOTL(\widetilde{S})$ or $FOTL$ when the vocabulary is of no importance. $FOTL(\widetilde{S})$ is defined by extending the language TCL of Section 3 as follows:

- A new sort $\mathcal{D}$ is introduced. The set of constants of sort $\mathcal{D}$ is *Dom* (the data domain introduced in Section 4). A countably infinite set of variables of sort $\mathcal{D}$ and an equality symbol $=_{\mathcal{D}}$ are also introduced.

- For each scheme $R(C_1, \ldots, C_m) \in \widetilde{S}$, an m-place predicate symbol R of sort $sort(C_1) \times \cdots \times sort(C_m)$ is introduced. These will be called *database* predicate symbols.

The well-formed formulas of $FOTL(\widetilde{S})$ are defined in the usual way. A many-sorted structure $(\mathcal{T}, Dom, \widetilde{R})$ will interpret symbols of $FOTL(\widetilde{S})$ as follows:

- To the symbols of $FOTL(\widetilde{S})$ which are also symbols of TCL, $(\mathcal{T}, Dom, \widetilde{R})$ makes the same assignments as the structure $\mathcal{T}$ defined in Section 3.
- To the sort $\mathcal{D}$, $(\mathcal{T}, Dom, \widetilde{R})$ assigns the set Dom.
- To every constant c of sort $\mathcal{D}$, $(\mathcal{T}, Dom, \widetilde{R})$ assigns $c \in Dom$.
- To the sequence $\widetilde{p} = (p_1, \ldots, p_n)$ of database predicate symbols, $(\mathcal{T}, Dom, \widetilde{R})$ assigns the sequence of relations $\widetilde{R} = (R_1, \ldots, R_n)$.

5.1 Relational Calculus with Temporal Constraints

The first declarative query language which we will define is relational calculus with temporal constraints. This query language is an instance of the constraint query language scheme of [33].

Let $\widetilde{S}$ be a database scheme and $R(C_1, \ldots, C_m)$ be a relation scheme. An expression over $\widetilde{S}$ of *relational calculus with temporal constraints* is

$$\{R(C_1, \ldots, C_m), x_1/s_1, \ldots, x_m/s_m \ : \ \phi(x_1, \ldots, x_m)\}$$

where s_i is the sort of C_i, ϕ is a well-formed formula of $FOTL(\widetilde{S})$ and $x_1, \ldots, x_m$ are the only free variables of ϕ.

Expressions of relational calculus with temporal constraints define functions from relational databases to relations as follows.

Definition 5.1 Let $\widetilde{S}$ be a database scheme and f be the following expression over $\widetilde{S}$ in relational calculus with temporal constraints:

$$\{R(C_1, \ldots, C_m), x_1/s_1, \ldots, x_m/s_m \ : \ \phi(x_1, \ldots, x_m)\}$$

Let $\widetilde{R}$ be a relational database over $\widetilde{S}$. The *value* of f on input database $\widetilde{R}$, denoted by $f(\widetilde{R})$, is the relation

$$\{(a_1, \ldots, a_m) \in dom(s_1) \times \cdots \times dom(s_m) : \ (\mathcal{T}, Dom, \widetilde{R}) \models \phi(a_1, \ldots, a_m)\}.$$

Expressions of relational calculus with temporal constraints can also be understod as defining functions from generalized relational databases to *relations* as follows.

Definition 5.2 Let $\widetilde{S}$ be a database scheme and f be the following expression over $\widetilde{S}$ in relational calculus with temporal constraints:

$$\{R(C_1, \ldots, C_m), x_1/s_1, \ldots, x_m/s_m \ : \ \phi(x_1, \ldots, x_m)\}$$

Let $\widetilde{R}$ be a generalized relational database over $\widetilde{S}$. The *value* of f on input database $\widetilde{R}$, denoted by $f(\widetilde{R})$, is the relation

$$\{(a_1, \ldots, a_m) \in dom(s_1) \times \cdots \times dom(s_m) : \ (\mathcal{T}, Dom, points(\widetilde{R})) \models \phi(a_1, \ldots, a_m)\}.$$

The above definition is somewhat problematic. The result of a query on a generalized relational database is defined to be an unrestricted relation. Intuitively we would like to guarantee *closure* as pointed out in [33]. In other words, given an expression f of relational calculus with temporal constraints, and a generalized relational database $\widetilde{R}$, is it possible to find a generalized relation which finitely represents $f(\widetilde{R})$? In Section 8, we show that this closure property can be guaranteed for *safe* expressions in relational calculus with temporal constraints.

Expressions of relational calculus with temporal constraints can also be used for querying temporal tables. To achieve this we have to resort to the set of unrestricted relations represented by the temporal tables. Similar definitions have appeared in [26, 23] in the case of conditional tables.

Definition 5.3 Let $\widetilde{S}$ be a database scheme and f be the following expression over $\widetilde{S}$ in relational calculus with temporal constraints:

$$\{T(C_1, \ldots, C_m), x_1/s_1, \ldots, x_m/s_m \ : \ \phi(x_1, \ldots, x_m)\}$$

Let $\widetilde{T}$ be a table database over $\widetilde{S}$. The *value* of f on input database $\widetilde{T}$, denoted by $f(\widetilde{T})$, is the set of temporal relations

$$\{f(\widetilde{R}) : \widetilde{R} \in rep(\widetilde{T})\}.$$

As can be seen from the above definition, the value of a relational calculus expression on a temporal table database can be an infinite set whose elements are infinite sets of tuples! In Section 8, we will show that safe expressions in relational calculus with temporal constraints can be evaluated in closed form. In other words, given a safe expression f of relational calculus with temporal constraints and a temporal table database $\widetilde{T}$, it is possible to find a temporal table which finitely represents $f(\widetilde{T})$.

5.2 Modal Relational Calculus with Temporal Constraints

As we explained in Section 2.3 modal queries are important in any indefinite information context. In this section, we extend the relational calculus with temporal constraints by introducing the modal operators of *possibility* and *certainty* [24]. Note that the use of these operators is restricted. Similar query languages have been investigated in [43, 42, 51].

Let $\widetilde{S}$ be a database scheme and $T(C_1, \ldots, C_m)$ be a scheme. An expression over $\widetilde{S}$ in *modal relational calculus with temporal constraints* is

$$\{T(C_1, \ldots, C_m), x_1/s_1, \ldots, x_m/s_m \ : \ \Diamond\ \phi(x_1, \ldots, x_m)\} \quad \text{or}$$

$$\{T(C_1, \ldots, C_m), x_1/s_1, \ldots, x_m/s_m \ : \ \Box\ \phi(x_1, \ldots, x_m)\}$$

where s_i is the sort of C_i, ϕ is a well-formed formula of $FOTL(\widetilde{S})$, $x_1, \ldots, x_m$ are the only free variables of ϕ, $\Box$ is the modal operator for *certainty* and $\Diamond$ is the modal operator for *possibility*. Expressions with the operator $\Box$ (resp. $\Diamond$) will be called $\Box$-expressions (resp. $\Diamond$-expressions).

Expressions of modal relational calculus with temporal constraints define functions from temporal table databases to *singleton sets* of temporal relations as follows.

Definition 5.4 Let $\widetilde{S}$ be a database scheme and f be the following expression over $\widetilde{S}$ in modal relational calculus with temporal constraints:

$$\{T(C_1, \ldots, C_m), x_1/s_1, \ldots, x_m/s_m \ : \ \Box\ \phi(x_1, \ldots, x_m)\}$$

Let $\widetilde{T}$ be a table database over $\widetilde{S}$. The value of f on input database $\widetilde{T}$, denoted by $f(\widetilde{T})$, is the following set which contains a *single* relation:

$$\{\ \{(a_1,\ldots,a_m) \in dom(s_1) \times \cdots \times dom(s_m) : \text{ for every relational database } \widetilde{R} \in sem(\widetilde{T}),\ (\mathcal{T}, Dom, \widetilde{R}) \models \phi(a_1,\ldots,a_m)\}\ \}$$

Definition 5.5 Let $\widetilde{S}$ be a database scheme and f be the following expression over $\widetilde{S}$ in modal relational calculus with temporal constraints:

$$\{T(C_1,\ldots,C_m), x_1/s_1,\ldots,x_m/s_m\ :\ \Diamond\ \phi(x_1,\ldots,x_m)\}$$

Let $\widetilde{T}$ be a table database over $\widetilde{S}$. The value of f on input database $\widetilde{T}$, denoted by $f(\widetilde{T})$, is the following set which contains a *single* relation:

$$\{\ \{(a_1,\ldots,a_m) \in dom(s_1) \times \cdots \times dom(s_m) : \text{ there exists a relational database } \widetilde{R} \in sem(\widetilde{T}) \text{ s.t. } (\mathcal{T}, Dom, \widetilde{R}) \models \phi(a_1,\ldots,a_m)\}\ \}$$

Section 8 demonstrates that safe expressions in modal relational calculus with temporal constraints can be evaluated in closed form. For every safe expression f in modal relational calculus with temporal constraints, and table database $\widetilde{T}$, it is possible to find a u-table which finitely represents $f(\widetilde{T})$.

Definition 5.6 Let f be an expression of modal relational calculus with temporal constraints. Let f' be the expression of relational calculus with temporal constraints which is obtained from f by discarding the modal operator. Then f and f' will be called *corresponding*.

The next lemma demonstrates an intuitive property of modal calculus expressions. If $\mathcal{S}$ is a set of sets then $\bigcap \mathcal{S}$ (resp. $\bigcup \mathcal{S}$) denotes the set $\{\cap_{s \in \mathcal{S}} s\}$ (resp. $\{\cup_{s \in \mathcal{S}} s\}$).

Lemma 5.1 *Let f be a $\Box$-expression (resp. $\Diamond$-expression) over $\widetilde{S}$ in modal relational calculus with temporal constraints. Let f' be the expression of relational calculus with temporal constraints which corresponds to f. Then for all temporal table databases $\widetilde{T}$ over $\widetilde{S}$, $f(\widetilde{T}) = \bigcap f'(\widetilde{T})$ (resp. $f(\widetilde{T}) = \bigcup f'(\widetilde{T})$).*

Proof: Let ϕ be the FOTL formula of f and f'. The following equivalences prove the lemma for $\Box$-expressions. The proof is similar for $\Diamond$-expressions.

$(a_1,\ldots,a_m) \in f(\widetilde{T})$ iff
for every relational database $\widetilde{R} \in sem(\widetilde{T})$, $(\mathcal{T}, Dom, \widetilde{R}) \models \phi(a_1,\ldots,a_m)$ iff
for every generalized relational database $\widetilde{R}' \in rep(\widetilde{T})$,
$(\mathcal{T}, Dom, points(\widetilde{R}')) \models \phi(a_1,\ldots,a_m)$ iff
for every generalized relational database $\widetilde{R}' \in rep(\widetilde{T})$, $(a_1,\ldots,a_m) \in f'(\widetilde{R}')$ iff
$(a_1,\ldots,a_m) \in \bigcap f'(\widetilde{T})$.

■

5.3 Examples

Let us now give some examples of queries in (modal) relational calculus with temporal constraints over the database of Section 2.3.

Example 5.1 Find the vehicles that are on service throughout the first hour of the afternoon.

This query can be expressed in relational calculus with temporal constraints as follows:

$$\{ANS(VEH), x/\mathcal{D} : \ (\exists i/\mathcal{I})(\exists c/\mathcal{D})(ON\_SERVICE(x, c, i) \ \wedge \ i_L \leq 12 \ \wedge \ i_R \geq 13)\}.$$

Example 5.2 Find all times during which it is certain that both vehicles V1 and V2 are on service.

This query can be expressed in modal relational calculus with temporal constraints as follows:

$$\{ANS(TIME), i/\mathcal{I} : \ \Box(\exists c1, c2/\mathcal{D})(ON\_SERVICE(V1, c1, i) \ \wedge \ ON\_SERVICE(V2, c2, i))\}.$$

6 Procedural Query Languages

In this section, we develop three procedural query languages: the *relational algebra* for temporal relations, the *generalized relational algebra* for generalized temporal relations and the *table algebra* for temporal tables. The relational algebra operations are defined exactly as in the case of finite relations. Most of the operations of the generalized relational algebra are extensions of similar operations of the standard relational algebra. There are also two extra operators useful for dealing with temporal entities in our model. The table algebra contains two additional operators $POSS$ and $CERT$.

6.1 The Relational Algebra and the Generalized Relational Algebra

The relational algebra operations can be defined exactly as in the case of finite relations (verbatim) [32]. Naturally the relational algebra is interesting only from a theoretical point of view because it operates on infinite relations.

We now introduce the generalized relational algebra which operates on finite representations of infinite relations. We initially give definitions for selection, projection, join, difference, union, rename, cartesian product and intersection. Similar operations have been defined in [31].

Selection. Let us first define *selection conditions* as follows:

1. A temporal constraint whose variables are temporal attributes is a *selection condition* (in particular an *atomic temporal selection condition.*)
2. A formula of the form $A = a, A \neq a, A = B$ or $A \neq B$, where A, B are data attributes and $a \in Dom$, is a *selection condition* (in particular an *atomic data selection condition.*)
3. If E_1 and E_2 are selection conditions then $E_1 \ \wedge \ E_2$ is a selection condition.

A selection condition E will usually be written as $E_\delta \ \wedge \ E_\tau$ where E_τ is a conjunction of temporal selection conditions and E_δ is a conjunction of data selection conditions.

Let R be a generalized relation on scheme S and $E = E_\delta \ \wedge \ E_\tau$ a selection condition. Then $\sigma_E(R)$ is defined as follows:

1. $sch(\sigma_E(R)) = sch(R)$.
2. $\sigma_E(R) = \{\sigma_E(t) : \ t \in R$ and $t(S_d)$ satisfies $E_\delta\}$

where $\sigma_E(t)(S) = t(S)$ and $\sigma_E(t)(CON) = t(CON) \ \wedge \ E_\tau(t)$. The constraint $E_\tau(t)$ is obtained from E_τ by substituting $t(A)$ for each temporal attribute A that appears in E_τ. For the data part of t satisfaction is defined as for the standard relational case.

Projection. If R is a generalized relation on scheme S and $X \subseteq S$ then $\pi_X(R)$ is defined as follows:

1. $sch(\pi_X(R)) = X$.

2. $\pi_X(R) = \{\pi_X(t) : \; t \in R\}$

where $\pi_X(t)$ is a tuple with scheme X and $\pi_X(t)(X) = t(X)$ and $\pi_X(t)(CON)$ is obtained from $t(CON)$ after all the variables $u \in V^u$ such that there exists $A \in S \setminus X$ and $t(A) = u$ are eliminated. If $X = \emptyset$ then for any tuple t, $t(X)$ is the empty tuple $()$.

Join. Let us assume that S_1 and S_2 are the schemes of generalized relations R_1 and R_2, X_t is the set of temporal attributes common to S_1 and S_2 and X_d is the set of data attributes common to S_1 and S_2. The join of R_1 and R_2 can now be defined as follows:

1. $sch(R_1 \bowtie R_2) = sch(R_1) \cup (sch(R_2) \setminus (X_t \cup X_d))$.

2. $R_1 \bowtie R_2 = \{t_1 \bowtie t_2 : \; t_1 \in R_1, \; t_2 \in R_2$ and $t_1(X_d) = t_2(X_d)\}$.

The join of two tuples $t_1 \bowtie t_2$ is defined by the following:

1. $(t_1 \bowtie t_2)(A) = t_1(A)$ if $A \in S_1$

2. $(t_1 \bowtie t_2)(A) = t_2(A)$ if $A \in S_2 \setminus (X_t \cup X_d)$

3. $(t_1 \bowtie t_2)(CON) = t_1(CON) \; \wedge \; t_2'(CON)$ where $t_2'(CON)$ is the same as $t_2(CON)$ but with each variable $t_2(A)$ substituted by $t_1(A)$ for each $A \in X_t$.

Difference. Let us take generalized relations R and W with scheme $S = S_t \cup S_d$. The difference of R and W can now be defined as follows:

1. $sch(R - W) = sch(R) = sch(W)$.

2. $R - W = \bigcup_{t \in R} (t_W)^{\uparrow}$ where $W = \{t_1, \cdots, t_n\}$ and $t_W = (((t - t_1) - t_2) - \cdots - t_n)$.

The difference of two tuples $t_1 - t_2$ is a *generalized relation* which is defined as follows:

1. $t_1 - t_2 = \{t_1\}$ if $t_2(S_d) \neq t_1(S_d)$.

2. $t_1 - t_2 = \{s : \; s(S_d) = t_1(S_d)$ and $s(S_t \cup \{CON\}) = \overline{t_2(S_t \cup \{CON\})} \cap t_1(S_t \cup \{CON\}) \; \}$ otherwise.

To understand the above formulas, we need the following clarifications. The tuple $\overline{t}$ is the complement of t. It is computed by negating the constraints and leaving the rest of the tuple the same. We will abuse our terminology and call $\overline{t}$ a tuple although its local constraint is a conjunction of temporal constraints and negated conjunctions of temporal constraints.

The intersection of two tuples $t_1 \cap t_2$ is defined as follows:

$$(t_1 \cap t_2)(S_t) = t_1(S_t) \;\; \text{and} \;\; (t_1 \cap t_2)(CON) = t_1(CON) \; \wedge \; t_2'(CON)$$

where $t_2'(CON)$ is the same as $t_2(CON)$ but with each variable $t_2(A)$ substituted by $t_1(A)$ for each $A \in S_t$.

The operation $R^{\uparrow}$ has the effect of *normalizing* the local constraints of generalized relation R in order to obtain a true generalized relation. This is done by the following three steps:

- Application of De Morgan's laws to transform the negated part of each local constraint of R (which was produced by the complementation operation) into a disjunction whose disjuncts are inequality constraints or conjunctions of equality constraints.

- Application of the law of associativity of conjunction with respect to disjunction to transform each local constraint of R into a disjunction of temporal constraints.

- Splitting of disjunctive constraints into different tuples with the same temporal part.

Union. Let R_1 and R_2 be generalized relations over scheme S. Then the union of R_1 and R_2 is defined as follows:

1. $sch(R_1 \cup R_2) = sch(R_1) = sch(R_2)$

2. $R_1 \cup R_2 = \{t : \; t \in R_1 \text{ or } t \in R_2\}$

Rename: Let R be a generalized relation over scheme $\{C_1, \ldots, C_n\}$. Let A be an attribute of U and $A, C_i \in U_D$ or $A, C_i \in U_P$ or $A, C_i \in U_I$. Then the renaming of attribute C_i to A is defined as follows:

- $sch(\theta_A^{C_i}(R)) = \{C_1, \ldots, C_{i-1}, A, C_{i+1}, \ldots, C_n\}$

- $\theta_A^{C_i}(R) = \{t : \; t \in R\}$

Cartesian Product, Intersection. The definitions of these two operations are omitted for brevity.

The operations defined so far were extensions of relational algebra operations to generalized relations. We now define two operations which are useful for the manipulation of intervals and points. For these definitions let R be a generalized relation over scheme $S = S_d \cup S_p \cup S_i$ where $S_d \subset U_D$, $S_p \subset U_P$ and $S_i \subset U_I$.

PointsToIntervals: Let Y be the string $L_1, R_1, \ldots, L_n, R_n \; : \; A_1, \ldots, A_n$ where $L_i, R_i \in S_p$, $A_i \in U_I$ $(i = 1, \ldots, n)$ and $\{A_1, \ldots, A_n\} \cap S = \emptyset$.

The operation $PointsToIntervals_Y$, which will be denoted by PTI_Y, creates a new interval attribute A_i from each ordered pair of point attributes L_i, R_i of input generalized relation R (for each $i = 1, \ldots, n$). Formally, $PTI_Y(R)$ is defined as follows:

1. $sch(PTI_Y(R)) = (sch(R) \setminus \{L_1, R_1, \ldots, L_n, R_n\}) \cup \{A_1, \ldots, A_n\}$

2. $PTI_Y(R) = \{pti(t) : \; t \in R\}$

where

1. $pti(t)(A) = t(A)$ if $A \in S_d \cup S_i \cup (S_p \setminus \{L_1, R_1, \ldots, L_n, R_n\})$

2. $pti(t)(A_i) = X^i$ where X^i is a new variable from V_u^i (for each $i = 1, \ldots, n$).

3. $pti(t)(CON) = t(CON)'$ where $t(CON)'$ is the same as $t(CON)$ but with point variables $t(L_i)$ and $t(R_i)$ substituted by X_L^i and X_R^i respectively (for each $i = 1, \ldots, n$).

IntervalsToPoints: Let Y be the string $A_1, \ldots, A_n \; : \; L_1, R_1, \ldots, L_n, R_n$ where $L_j, R_j \in U_P$, $A_j \in S_i$ $(j = 1, \ldots, n)$ and $\{L_1, R_1, \ldots, L_n, R_n\} \cap S = \emptyset$.

The operation $IntervalsToPoints_Y$, which will be denoted by ITP_Y, creates two point attributes L_i, R_i from each interval attribute A_i of input generalized relation R (for each $i = 1, \ldots, n$). Formally, $ITP_Y(R)$ is defined as follows:

1. $sch(ITP_Y(R)) = (sch(R) \setminus \{A_1, \ldots, A_n\}) \cup \{L_1, R_1, \ldots, L_n, R_n\}$

2. $ITP_Y(R) = \{itp(t) : \; t \in R\}$

where

1. $itp(t)(A) = t(A)$ if $A \in S_d \cup S_p \cup (S_i \setminus \{A_1, \ldots, A_n\})$.
2. $itp(t)(L_i) = X_{L_i}$ and $itp(t)(R_i) = X_{R_i}$, where X_{L_i} and X_{R_i} are brand new variables from V_u^p (for each $i = 1, \ldots, n$).
3. $itp(t)(CON) = t(CON)' \wedge \bigwedge_{i=1}^{n} X_{L_i} < X_{R_i}$ where $t(CON)'$ is the same as $t(CON)$ but with terms $t(A_i)_L$ and $t(A_i)_R$ substituted by X_{L_i} and X_{R_i} respectively (for each $i = 1, \ldots, n$).

In some cases, *constant generalized relations* are important. We will follow [44] and denote constant generalized relations as sets of tuples with explicit attribute naming. For example, if $A \in U_D, B \in U_P, a_1, a_2 \in Dom$ and $t_1, t_2 \in V_u^p$ then

$$\{(A : a_1, B : t_1, CON : t_1 \leq 5),\ (A : a_2, B : t_2, CON : true)\}$$

is a constant generalized relation.

Definition 6.1 A *generalized relational algebra expression* over scheme $\widetilde{S}$ is any well-formed expression built-up from constant generalized relations, generalized relation scheme names from $\widetilde{S}$ and the above operators. ■

Generalized relational algebra expressions define functions from generalized relational databases to generalized relations as follows. If e is a generalized relational algebra expression over $\widetilde{S}$ and $\widetilde{R}$ is a generalized relational database over $\widetilde{S}$, then $e(\widetilde{R})$ (the result of applying e to $\widetilde{R}$) denotes the generalized relation R' obtained by substituting relations from $\widetilde{R}$ for relation names in e and evaluating the expression in a *bottom-up* fashion using the above definitions. The scheme $sch(R') = sch(e(\widetilde{R}))$ is also determined by the previous definitions.

Definition 6.2 Let e_1, e_2 be generalized relational algebra expressions over $\widetilde{S}$ such that $sch(e_1) = sch(e_2)$. The expressions e_1 and e_2 are called *equivalent* iff for all generalized relational databases $\widetilde{R}$ over $\widetilde{S}$, $points(e_1(\widetilde{R})) = points(e_2(\widetilde{R}))$. ■

The following proposition presents a subset of generalized relational algebra operations which has the same expressive power as the full algebra.

Proposition 6.1 *Let e_1 be an expression of the generalized relational algebra over $\widetilde{S}$. Then there exists an equivalent generalized relational algebra expression e_2 over the same scheme which uses only constant generalized relations, selection, projection, union, join, difference and rename. More precisely, (i) constant generalized relations can have a single tuple with a single attribute and local constraint "true", and (ii) selections can involve a single atomic condition (data or temporal).*

Proof: The proof of this theorem is an easy generalization of theorem 3.1 of [44] which covers the relational case. For the operations ITP and PTI, let us simply observe that the following equations hold:

1. $points(PTI_{B,E:A}(R1)) = points(\pi_{A,S'}(\sigma_{A_L=B \wedge A_R=E}(R1 \bowtie \{(A : i, CON : true)\})))$ where $R1$ is a generalized relation over S and $S' = S \setminus \{B, E\}$.
2. $points(ITP_{A:B,E}(R)) = points(\pi_{B,E,S'}(\sigma_{B=A_L \wedge E=A_R}(R \bowtie \{(B : t_1, E : t_2, CON : true)\})))$ where R is a generalized relation over S and $S' = S \setminus \{A\}$.

■

6.2 The Table Algebra

Let us now turn our attention to the table algebra. The operations of the table algebra take as input one (or two) tables associated with a common global constraint and return a table associated with the *same* global constraint.

The table algebra contains an operator for every generalized relational algebra operator defined in the previous section. The definitions of these operations were originally defined in [36]. They are the same as for generalized relational algebra operations (verbatim !). Therefore these operations treat e-variables as uninterpreted parameters. However, the table algebra also includes two additional operators, $POSS$ and $CERT$, which take a more active stand towards e-variables.

Given a table $\widetilde{T}$, the expression $POSS(\widetilde{T})$ evaluates to a u-table which finitely represents the set of all tuples contained in *any* relation of $sem(\widetilde{T})$. The expression $CERT(\widetilde{T})$ evaluates to a u-table which finitely represents the set of all tuples contained in *every* relation of $sem(\widetilde{T})$.

Possibility. Let T be a table on scheme S. Then $POSS(T)$ is a *u-table* defined as follows:

1. $sch(POSS(T)) = sch(T)$
2. $POSS(T) = \{poss(t) : t \in T\}$

For each tuple t on scheme S, $poss(t)$ is a tuple on scheme S such that $poss(t)(S) = t(S)$ and $poss(t)(CON) = \psi$ where ψ is obtained by eliminating all e-variables from the conjunction of constraints $G(T) \wedge t(CON)$.

Certainty. Let T be a table on scheme S. Then $CERT(T)$ is a *u-table* defined as follows:

1. $sch(CERT(T)) = sch(T)$
2. $CERT(T) = \{cert(t) : t \in T^{\downarrow}\}^{\uparrow}$

For each tuple t on scheme S, $cert(t)$ is a tuple on scheme S such that $cert(t)(S) = t(S)$ and $cert(t)(CON) = \neg\psi$ where ψ is obtained by eliminating all e-variables from the conjunction of constraints $G(T) \wedge \neg t(CON)$. Eliminating variables from $G(T) \wedge \neg t(CON)$ is done in two steps. First, $G(T) \wedge \neg t(CON)$ is transformed into disjunctive normal form. Then variable elimination takes place independently in each disjunct.

The operation $R^{\downarrow}$ has the effect of *denormalizing* generalized relation R. This is achieved by collecting tuples $\{t_1, \ldots, t_k\}$ of T with the same data part $(a_1, \ldots, a_{|S_d|})$ into a single tuple t' on scheme S such that $t'(S_d) = (a_1, \ldots, a_{|S_d|})$, $t'(S_t) = (x_1, \ldots, x_{|S_t|})$ and $t'(CON) = t'_1(CON) \vee \cdots \vee t'_k(CON)$. In the new tuple t' u-variables have been standardized apart: $x_1, \ldots, x_{|S_t|}$ are brand new u-variables, and for $1 \leq i \leq k$, $t'_i(CON)$ is the same as $t_i(CON)$ except that $t(X)$ has been substituted by $t'(X)$ for each $X \in S_t$.

The following lemma gives an intuitive property of $POSS$ and $CERT$.

Lemma 6.1 *Let e be a pure table algebra expression over scheme $\widetilde{S}$. Then for all table databases $\widetilde{T}$ over $\widetilde{S}$*

$$sem(CERT(e(\widetilde{T}))) = \bigcap sem(e(\widetilde{T})) \text{ and } sem(POSS(e(\widetilde{T}))) = \bigcup sem(e(\widetilde{T})).$$

Proof: Let $W = e(\widetilde{T})$ be the result of applying e to $\widetilde{T}$, and let $sch(W) = S$. We will show that $sem(CERT(W)) = \bigcap sem(W)$.

Let δ be a sequence of elements of Dom, τ a sequence of elements of $\mathbb{Q} \cup Int(\mathbb{Q})$ and χ a sequence of elements of V^u. A relational tuple t over $\widetilde{S}$ such that $t(S_d) = \delta$ and $t(S_t) = \tau$ will be denoted by (δ, τ). Similarly, a table tuple t over $\widetilde{S}$ such that $t(S_d) = \delta$ and $t(S_t) = \chi$ will be denoted by (δ, χ).

Let us assume $sem(CERT(W)) = \{R_1\}$ and $\bigcap sem(W) = \{R_2\}$ where R_1 and R_2 are relations over $\widetilde{S}$. Initially, we will show that $R_1 \subseteq R_2$.

Let us assume $(\delta, \tau) \in R_1$. Then there exists a tuple $t = (\delta, \chi) \in CERT(W)^{\downarrow}$ such that $\mathcal{T} \models t(CON)[\chi \leftarrow \tau]$. Therefore $\mathcal{T} \not\models \neg t(CON)[\chi \leftarrow \tau]$. Notice that $\neg t(CON)$ is obtained by eliminating all e-variables from $G(W) \ \wedge \ \neg t'(CON)$ where t' is the tuple of $W^{\downarrow}$ such that $t'(S_d) = \delta$.

Now assume that $(\delta, \tau) \notin R_2$. Therefore there exists a valuation $v \in Val^e$ such that $\mathcal{T} \models G(W)[v]$ and $\mathcal{T} \not\models t'(CON)[v; \chi \leftarrow \tau]$. In this case, $\mathcal{T} \models G(W)[v]$ and $\mathcal{T} \models \neg t'(CON)[v; \chi \leftarrow \tau]$ thus $\mathcal{T} \models \neg t(CON)[\chi \leftarrow \tau]$. Contradiction! Then it must be $(\delta, \tau) \in R_2$.

Let us now show $R_2 \subseteq R_1$. Assume $(\delta, \tau) \in R_2$. Then there exists a tuple $t = (\delta, \chi) \in W^{\downarrow}$ such that, for all valuations $v \in Val^e$, if $\mathcal{T} \models G(W)[v]$ then $\mathcal{T} \models t(CON)[v; \chi \leftarrow \tau]$.

Now assume $(\delta, \tau) \notin R_1$. If t' is the tuple of $CERT(T)^{\downarrow}$ such that $t'(S_d) = \delta$ and $t'(S_t) = \chi$ then $\mathcal{T} \not\models t'(CON)[\chi \leftarrow \tau]$. Therefore $\mathcal{T} \models \neg t'(CON)[\chi \leftarrow \tau]$. The formula $\neg t'(CON)$ is obtained after all e-variables are eliminated from $G(T) \ \wedge \ \neg t(CON)$. Therefore valuation $\chi \leftarrow \tau$ can be extended to a new valuation $v'; \chi \leftarrow \tau$ such that $v' \in Val^e$ and $\mathcal{T} \models G(T) \ \wedge \ \neg t(CON)[v'; \chi \leftarrow \tau]$. But this contradicts the conclusion of the previous paragraph. Therefore $(\delta, \tau) \in R_1$.

The proof for $POSS$ is similar. ■

Let us now define table algebra expressions.

Definition 6.3 A *pure table algebra expression* over scheme $\widetilde{S}$ is any well-formed expression built from constant u-tables, table scheme names from $\widetilde{S}$ and the above operators excluding $POSS$ and $CERT$. A *table algebra expression* is a pure table algebra expression, or an expression of the form $CERT(g)$ or $POSS(g)$ where g is a pure table algebra expression. Expressions of the form $CERT(g)$ or $POSS(g)$ are called $CERT$*-expressions* or $POSS$*-expressions* respectively. ■

Table algebra expressions define functions from table databases to tables. The result of applying an expression e to a table database $\widetilde{T}$ is defined as in the generalized relational expression case. Let us simply stress that $G(e(\widetilde{T})) = G(\widetilde{T})$ for all table databases $\widetilde{T}$ and expressions e over $\widetilde{S}$.

Definition 6.4 Let e_1 and e_2 be table algebra expressions over scheme $\widetilde{S}$ such that $sch(e_1) = sch(e_2)$. Expressions e_1 and e_2 are called *equivalent* iff for all table databases $\widetilde{T}$ over $\widetilde{S}$, $sem(e_1(\widetilde{T})) = sem(e_2(\widetilde{T}))$. ■

The following proposition presents a subset of table algebra operations which has the same expressive power as the full algebra.

Proposition 6.2 *Let e_1 be an expression of the table algebra over $\widetilde{S}$. Then there exists an equivalent table algebra expression e_2 over $\widetilde{S}$ which uses only constant u-tables, selection, projection, union, join, difference, rename and $POSS$ (or $CERT$). More precisely, (i) constant u-tables can have a single tuple with a single attribute and local constraint "true", and (ii) selections can involve a single atomic condition (data or temporal).*

Proof: Initially assume that e_1 is pure. Let e_1' be the generalized relational algebra expression corresponding to e_1. Then there exists an equivalent generalized relational algebra expression e_2' which uses only constant generalized relations, selection, projection, union, join, difference and rename. We will take e_2 to be the pure table algebra expression corresponding to e'_2. The equivalence of e_1 and e_2 is demonstrated by the following:

$$sem(e_1(\widetilde{T})) = \{points(e'_1(\widetilde{R})) : \widetilde{R} \in rep(\widetilde{T})\} = \{points(e'_2(\widetilde{R})) : \widetilde{R} \in rep(\widetilde{T})\} = sem(e_2(\widetilde{T}))$$

The extension to $CERT$-expressions and $POSS$-expressions is trivial. ■

6.3 Examples

In this section, we give examples of queries using the algebraic operations defined above. We will refer to the database of Figure 3.

Example 6.1 Find the vehicles that are on service throughout the first hour of the afternoon.

This query can be answered by evaluating the expression

$$\pi_{VEH}(\sigma_{TIME_L \leq 12 \,\wedge\, 13 \leq TIME_R}(ON\_SERVICE))$$

The following table is the result of the inside selection operation.

VEH	STATION	TIME	CON
V1	Athens	$i1$	$\omega 1_L \leq i1_L,\ i1_R \leq \omega 1_R,\ i1_L \leq 12,\ 13 \leq i1_R$
V2	Piraeus	$i2$	$\omega 2_L \leq i2_L,\ i2_R \leq \omega 2_R,\ i2_L \leq 12,\ 13 \leq i2_R$

To execute the projection operation, we need to eliminate the variables $i1$ and $i2$ from the local constraints. After translating the temporal constraints of the first tuple into point constraints, we are left with the following constraints:[10]

$$B_{i1} < E_{i1},\ B_{\omega 1} < E_{\omega 1},\ B_{\omega 1} \leq B_{i1},\ E_{i1} \leq E_{\omega 1},\ B_{i1} \leq 12,\ 13 \leq E_{i1}.$$

Eliminating B_{i1} and E_{i1} gives: $B_{\omega 1} \leq E_{\omega 1}$, $13 \leq E_{\omega 1}$, $B_{\omega 1} \leq 12$, $B_{\omega 1} < E_{\omega 1}$. Thus when the translation back to temporal constraints is performed, we get $\omega 1_L \leq 12,\ 13 \leq \omega 1_R$. Similarly, $\omega 2_L \leq 12,\ 13 \leq \omega 2_R$ is obtained from the second tuple. Therefore, we arrive at the following table:

VEH	CON
V1	$\omega 1_L \leq 12,\ 13 \leq \omega 1_R$
V2	$\omega 2_L \leq 12,\ 13 \leq \omega 2_R$

Now observe that the constraint of the first tuple and the first conjunct of the constraint of the second tuple are logically implied by the global constraint G. Thus we can simplify the constraints and arrive at the table ANS1 of Section 2.3. In general, simplifications of the same nature are achieved by computing the *canonical form* of a table. This matter is further discussed in [35].

Example 6.2 Find all times during which it is certain that both vehicles V1 and V2 are on service.

This query can be answered by evaluating the expression

$$CERT(\pi_{TIME}(\sigma_{VEH=V1}(ON\_SERVICE)) \bowtie \pi_{TIME}(\sigma_{VEH=V2}(ON\_SERVICE))).$$

The evaluation of the argument of $CERT$ results in the following table:

[10] For each interval i, terms i_L and i_R are translated into a point variables B_i and E_i respectively.

TIME	CON
i	$\omega 1_L \leq i_L,\ i_R \leq \omega 1_R,\ \omega 2_L \leq i_L,\ i_R \leq \omega 2_R$

To proceed with the evaluation of $CERT$, we have to eliminate all e-variables from

$$G' \wedge (i1_L < i1_R \vee i1_L < \omega 1_L \vee i1_R > \omega 1_R \vee i1_L < \omega 2_L \vee i1_R > \omega 2_R)$$

where G' is the conjunction of G and the constraints in the set

$$\{i_L < i_R :\ i \in V_i^e \text{ and } i \text{ appears in } G\}.$$

Transfoming the above constraint in disjunctive normal form and eliminating e-variables results in $i_L < 9 \vee i_R > 14 \vee i_L < 10.5 \vee i_R > 12$ or equivalently $i_L < 10.5 \vee i_R > 12$. Thus the answer is the following table ANS5.

ANS5

TIME	CON
i	$i_L \geq 10.5,\ i_R \leq 12$

Example 6.3 Find all times during which it is certain that vehicle V1 is on service in Athens but vehicle V2 is not.

This query can be answered by evaluating the following expression:

$$CERT(\pi_{TIME}(\sigma_{VEH=V1 \wedge CITY=Athens}(ON\_SERVICE)) \\ -\pi_{TIME}(\sigma_{VEH=V2 \wedge CITY=Athens}(ON\_SERVICE)))$$

Let T_1 and T_2 denote the first and the second operand of the above difference operator. Let us compute T_1 and T_2 first:

T_1

TIME	CON
$i1$	$\omega 1_L \leq i1_L,\ i1_R \leq \omega 1_R$

T_2

TIME	CON
$i2$	$\omega 2_L \leq i2_L,\ i2_R \leq \omega 2_R$

Since T_1 and T_2 have only one tuple each, we simply compute the difference of these two tuples s_1 and s_2. Observe that $\overline{s_2}(TIME) = t_2$, $\overline{s_2}(CON) = \neg(\omega 2_L \leq i2_L \wedge i2_R \leq \omega 2_R)$ therefore $(s_1 - s_2)(TIME) = i_1$ and $(s_1 - s_2)(CON) = \omega 1_L \leq i1_L \wedge i1_R \leq \omega 1_R \wedge (\omega 2_L > i2_L \vee i2_R > \omega 2_R)$.

After normalizing we get the following table $T_1 - T_2$:

$T_1 - T_2$

TIME	CON
$i1$	$\omega 1_L \leq i1_L,\ i1_R \leq \omega 1_R,\ \omega 2_L > i2_L$
$i3$	$\omega 1_L \leq i3_L,\ i3_R \leq \omega 1_R,\ i3_R > \omega 2_R$

Now we have to compute $CERT(T_1 - T_2)$. First we compute the table $(T_1 - T_2)^{\downarrow}$:

$(T_1 - T_2)^{\downarrow}$

TIME	CON
i	$(\omega 1_L \leq i_L \wedge i_R \leq \omega 1_R \wedge \omega 2_L > i_L) \vee$ $(\omega 1_L \leq i_L \wedge i_R \leq \omega 1_R \wedge i_R > \omega 2_R)$

Now we have to eliminate all e-variables from the following constraint

$G' \wedge \neg(\omega1_L \leq i_L \wedge i_R \leq \omega1_R \wedge \omega2_L > i_L \wedge i_L < i_R) \wedge \neg(\omega1_L \leq i_L \wedge i_R \leq \omega1_R \wedge i_R > \omega2_R \wedge i_L < i_R)$.

This elimination results in the constraint $i_L < 9 \ \vee \ i_R > 10$. Finally, the answer is given by the following table ANS6.

ANS6

TIME	CON
i	$i_L \geq 9,\ i_R \leq 10$

Example 6.4 Find all pairs of vehicles SV and AV such that it is possible that SV is on service in city CITY throughout interval SVT and AV arrives at CITY at some time point AVT during SVT. Output SV, CITY, SVT, AV and AVT.

The algebraic expression for this query is a bit complex so it will be built in steps. Let f be

$$\theta^{VEH}_{SV}(\theta^{TIME}_{SVT}(ON\_SERVICE)) \bowtie \theta^{TO}_{CITY}(\theta^{VEH}_{AV}(\theta^{TIME}_{ITRIP}(TRIP))),$$

g be $\sigma_{SV \neq AV \ \wedge \ SVT_L \leq ITRIP_R \ \wedge \ ITRIP_R \leq SVT_R}(f)$ and h be $\pi_{SV,CITY,SVT,AV,AVT}(ITP_{ITRIP:DT,AVT}(g))$. Then the algebraic expression for this query is $POSS(h)$.

The table corresponding to g is

SV	CITY	SVT	AV	FROM	ITRIP	CON
V1	Athens	$i1$	V3	Patra	$i3$	$\omega1_L \leq i1_L,\ i1_R \leq \omega1_R,\ i3_L = 9,$ $i3_R = \nu,\ i1_L \leq i3_R,\ i3_R \leq i1_R$
V2	Athens	$i2$	V3	Patra	$i4$	$\omega2_L \leq i2_L,\ i2_R \leq \omega2_R,\ i4_L = 9,$ $i4_R = \nu,\ i2_L \leq i4_R,\ i4_R \leq i2_R$

The table corresponding to $ITP_{ITRIP:DT,AVT}(g)$ is

SV	CITY	SVT	AV	FROM	DT	AVT	CON
V1	Athens	$i1$	V3	Patra	$p3$	$p1$	$\omega1_L \leq i1_L,\ i1_R \leq \omega1_R,\ p3 = 9,\ p1 = \nu,$ $i1_L \leq p1,\ p1 \leq i1_R,\ p1 < p2$
V2	Athens	$i2$	V3	Patra	$p4$	$p2$	$\omega2_L \leq i2_L,\ i2_R \leq \omega2_R,\ p4 = 9,\ p2 = \nu,$ $i2_L \leq p2,\ p2 \leq i2_R,\ p1 < p2$

Finally, the table for $POSS(h)$ is the following table ANS4.

ANS4

SV	CITY	SVT	AV	AVT	CON
V1	Athens	$i1$	V3	$p1$	$i1_L = 9,\ i1_R \leq 14,\ i1_L \leq p1 \leq i1_R,\ 12 \leq p1 \leq 13$
V2	Athens	$i2$	V3	$p2$	$i2_L \geq 10,\ i2_R \leq 14.5,\ i2_L \leq p2 \leq i2_R,\ 12 \leq p2 \leq 13$

6.4 Discussion

Let us now explain, as promised in Section 3, why using interval variables is significant. Interval variables are natural "higher-order" primitives which allow for more intuitive representation, and can possibly be exploited for more efficient query evaluation strategies.[11] The implementation

[11] The same argument in favour of "higher-order" variables has previously been made by [33] in a more general setting.

of our model will also allow all (or most) of the natural interval-to-interval and point-to-interval predicates of [2, 47]. If we had not introduced interval variables explicitly, facts true over intervals of time could have been represented in the following two ways: (i) using non-atomic attribute values (e.g., an interval variable will be represented by a term like $\langle p1, p2\rangle$ where p_1 and p_2 represent the endpoints of the interval), or (ii) using two different attributes for modeling the endpoints of an interval (this approach is taken by [31]). Option (i) requires new algebra operators (e.g., ITP and PTI) for going from interval data to point data and vice versa. These operations are *not* necessary in our model (proposition 6.1) although some users might prefer to use them. Option (ii) does not require any extra operators but this leaves the burden of providing interval interpretations for point data to the user. This disadvantage combined with the need for more efficient query evaluation has lead the authors of [31] to turn to interval-based representations in their more recent papers [61, 50].

7 On the Semantics of Algebraic Query Evaluation

Let $\widetilde{R}$ be a generalized relational database, e a generalized relational algebra expression and $e1$ its corresponding relational algebra expression. Recall that a generalized relation $\widetilde{R}$ is a finite representation of the unrestricted set of tuples $points(\widetilde{R})$. The following theorem shows that when we evaluate e on $\widetilde{R}$, we essentially evaluate $e1$ on the possibly infinite relation $points(\widetilde{R})$.

Theorem 7.1 *Let $\widetilde{R}$ be a generalized relational database over scheme $\widetilde{S}$, e a generalized relational algebra expression over $\widetilde{S}$ and $e1$ be its corresponding relational algebra expression. Then $points(e(\widetilde{R})) = e1(points(\widetilde{R}))$.*

Proof: Let R be a generalized relation and op a generalized relational algebra operation in the restricted set of operations defined in proposition 6.1. We will show that $points(op(R)) = op(points(R))$. Then the theorem follows easily. A detailed proof will be given only for projection and selection. The proofs for the rest of the operations are only sketched. Whenever it is not stated explicitly, relations are assumed to have scheme S.

Projection. Let us first prove that $points(\pi_X(R)) \subseteq \pi_X(points(R))$. Assume that $t \in points(\pi_X(T))$. Equivalently, there exists a tuple $t' \in \pi_X(R)$ and a valuation $u \in Val^u$ such that $t = u(t')$ and $\mathcal{T} \models t'(CON)[u]$. The fact $t' \in \pi_X(R)$ implies that there exists a tuple $t'' \in R$ such that $t''(X) = t'$ and $t'(CON)$ is obtained by elimination of the u-variables in $t''(S \setminus X)$ from $t''(CON)$. Let $x_1, \ldots, x_n$ be the variables appearing in $t'(X)$. Since $\mathcal{T} \models t'(CON)[u]$ then $(u(x_1), \ldots, u(x_n))$ is a solution of $t'(CON)$ thus it can be extended to a solution of $t''(CON)$ (definition 3.4). This solution defines a valuation $u' \in Val^u$ which is an extension of u such that $\mathcal{T} \models t''(CON))[u']$ and $u'(t''(X)) = u(t'(X)) = t$. Equivalently, there exists a relation tuple $s = u'(t'') \in points(R)$ and $s(X) = t$, therefore $t \in \pi_X(points(R))$.

Let us now prove that $\pi_X(points(R)) \subseteq points(\pi_X(R))$. Assume that $t \in \pi_X(points(R))$. Equivalently, there exists a tuple $t' \in points(R)$ and $t = t'(X)$. The fact that $t' \in points(R)$ implies that there exists a tuple $t'' \in R$ and a valuation $u \in Val^u$ such that $\mathcal{T} \models t''(CON)[u]$ and $t' = u(t''(S))$. We can now see that there is a tuple $s = t''(X) \in \pi_X(R)$ and valuation $u' \in Val^u(X)$ equal to the restriction of u on the set of variables in $t''(X)$ such that $u'(s) = u'(t''(X)) = u(t''(X)) = t'(X) = t$. In addition, $\mathcal{T} \models s(CON)[u]$ since $s(CON)$ is the same as $t''(CON)$ after the variables in $S \setminus X$ have been eliminated. In other words, $t \in points(\pi_X(R))$.

Selection. We will only show that $points(\sigma_E(R)) \subseteq \sigma_E(points(R))$ where E is an atomic selection condition (data or temporal). The inverse inclusion can be shown in a similar way. Let us assume that $t \in points(\sigma_E(R))$. Equivalently, there exists a tuple $t' \in \sigma_E(R)$ and a valuation

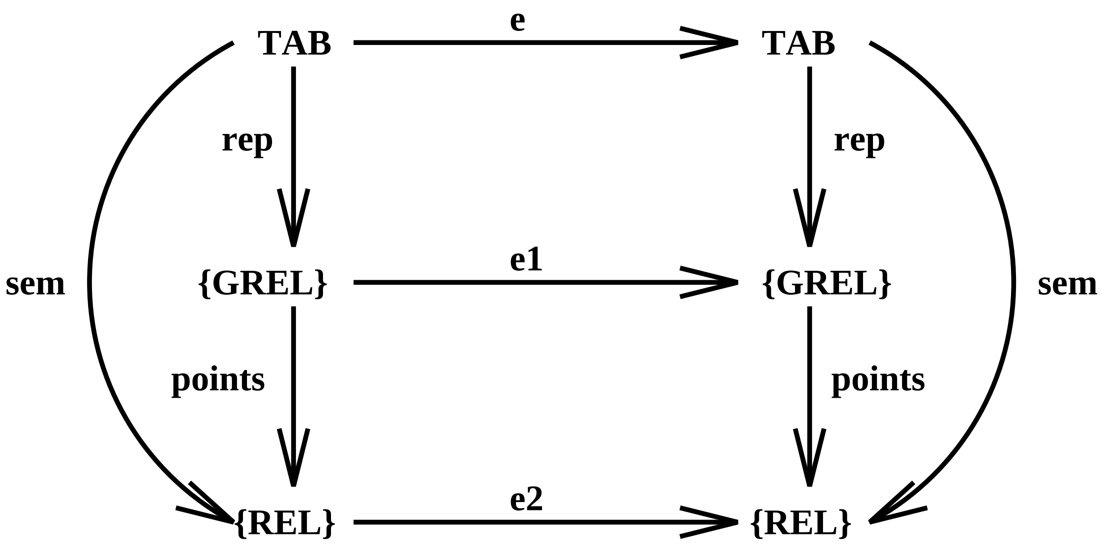

Figure 4: Relating the three algebras

$u \in Val^u$ such that $t = u(t'(S))$ and $\mathcal{T} \models t'(CON)[u]$. The fact $t' \in \sigma_E(R)$ implies that there exists a tuple $t'' \in R$ such that $t'(S) = t''(S)$. If E is a data selection condition then $t'(CON)$ is the same as $t''(CON)$. If E is a temporal selection condition then $t'(CON)$ is $t''(CON) \wedge E(t'')$ where $E(t'')$ is obtained from E by substituting $t''(A)$ for each attribute A that appears in E. In other words, there exists a tuple $s = u(t'') \in points(R)$ such that $t = u(t'(S)) = u(t''(S)) = s$ and $E(t)$ evaluates to true. Therefore $t \in \sigma_E(points(R))$.

Join. Assume R_1 and R_2 have schemes S_1 and S_2 respectively. We want to prove that $points(R_1 \bowtie R_2) = points(R_1) \bowtie points(R_2)$. Let us assume that $t \in points(R_1 \bowtie R_2)$. Equivalently, there exists a tuple $t' \in R_1 \bowtie R_2$ and a valuation $u \in Val^u$ and $t = u(t'(S_1 \cup S_2))$ and $\mathcal{T} \models t'(CON)[u]$. In other words $u(t'(S_1)) \in points(R_1)$ and $u(t'(S_2)) \in points(R_2)$. Therefore $t \in points(R_1) \bowtie points(R_2)$.

Difference. We want to prove that $points(R_1 - R_2) = points(R_1) - points(R_2)$. Let us assume that $t \in points(R_1 - R_2)$. Equivalently, there exists tuple t' and valuation u such that $t' \in R_1 - R_2$, $u \in Val^u$, $t = u(t'(S))$ and $\mathcal{T} \models t'(CON)[u]$. In other words, $u(t') \in points(R_1)$ and $u(t') \notin points(R_2)$. Therefore $t \in points(R_1) - points(R_2)$.

Rename, Union. The proof in this case is trivial. ∎

Let us now assume that $\widetilde{T}$ is a table database, e a pure table algebra expression and $e1$ its corresponding relational algebra expression. Recall that the semantic function $sem(\widetilde{T})$ returns all the "possible worlds" represented by $\widetilde{T}$. When we evaluate e on table database $\widetilde{T}$ using the operations defined above, we essentially evaluate the corresponding relational query on each possible world in $sem(\widetilde{T})$. As discussed in [25], an extension of a relational algebra expression $e1$ to an operation e for a table-like representation of indefinite information can claim to be "faithful to the underlying semantics" if and only if for every table database $\widetilde{T}$,

$$sem(e(\widetilde{T})) = e1(sem(\widetilde{T})) = \{e1(w) : \ w \in sem(\widetilde{T})\}.$$

Equivalently, one would like to guarantee that there is always a table T_1 such that $sem(T_1) = e1(sem(\widetilde{T}))$. The following theorem demonstrates that table databases satisfy this form of algebraic completeness.

Theorem 7.2 *Let $\widetilde{T}$ be a table database over scheme $\widetilde{S}$, e a pure table algebra expression over $\widetilde{S}$, $e1$ its corresponding generalized relational algebra expression and $e2$ its corresponding relational algebra expression. Then $rep(e(\widetilde{T}))$ is equivalent to $e1(rep(\widetilde{T}))$ and $sem(e(\widetilde{T})) = e2(sem(\widetilde{T}))$.*

Proof: We will only prove the second part of the theorem. The proof of the first part is similar. From the definition of sem, $sem(e(\widetilde{T})) = \{points(\widetilde{R}_1) : \widetilde{R}_1 \in rep(e(\widetilde{T}))\}$. In addition

$$\begin{aligned} rep(e(\widetilde{T})) = \{\widetilde{R}_2 :\ & \text{there exists } v \in Val^e \text{ and } \mathcal{T} \models G(e(\widetilde{T}))[v] \text{ and } \widetilde{R}_2 = v(e(\widetilde{T}))\} = \\ & \{v(e(\widetilde{T})) : \text{there exists } v \in Val^e \text{ and } \mathcal{T} \models G(\widetilde{T})[v]\}. \end{aligned}$$

Therefore $sem(e(\widetilde{T})) = \{points(v(e(\widetilde{T}))) : \text{there exists } v \in Val^e \text{ and } \mathcal{T} \models G(\widetilde{T})[v]\}$.

When an expression of the pure table algebra is evaluated, using the definitions in Section 6, the e-variables are treated as uninterpreted parameters (i.e., exactly like constants). Thus $v(e(\widetilde{T})) = e2(v(\widetilde{T}))$ for all table databases $\widetilde{T}$ over $\widetilde{S}$. Finally,

$$\begin{aligned} sem(e(\widetilde{T})) &= \{points(e2(v(\widetilde{T}))) : \text{there exists } v \in Val^e \text{ and } \mathcal{T} \models G(\widetilde{T})[v]\} \\ &= \{points(e2(\widetilde{R}')) : \text{there exists } v \in Val^e \text{ and } \mathcal{T} \models G(\widetilde{T})[v] \text{ and } \widetilde{R}' = v(\widetilde{T})\} \\ &= \{points(e2(\widetilde{R}')) : \widetilde{R}' \in rep(\widetilde{T})\} = \{e2(points(\widetilde{R}')) : \widetilde{R}' \in rep(\widetilde{T})\} \\ &= e2(sem(\widetilde{T})) \end{aligned}$$

■

The above theorems are summarized graphically in the commutative diagram of Figure 4.

8 Translating Safe Calculus Expressions into Algebraic Expressions

This section introduces the class of *safe* expressions in (modal) relational calculus with temporal constraints. Safe expressions can be evaluated bottom-up in closed form on generalized relational databases and table databases. This result is shown by exhibiting a translation of safe calculus expressions into algebraic expressions. This translation preserves the semantics of safe expressions thus it enables us to answer a safe calculus query by evaluating an equivalent algebraic query. Algebraic query evaluation can be done *bottom-up* and the answer is obtained *in closed form*. Thus safe expressions satisfy the requirements of the constraint query language framework of [33].

Our definition of safety is syntactic and is based on Ullman's definition of safe domain relational calculus [63]. We conjecture that the results of this section still hold even if we consider a larger class of "safe" calculus formulas e.g., the formulas considered by [67].

Definition 8.1 An expression of (modal) relational calculus with temporal constraints is called *safe* if its FOTL formula ϕ satisfies the following:

1. There are no uses of quantifier $\forall$.
2. If $\phi_1 \vee \phi_2$ is a subformula of ϕ then ϕ_1 and ϕ_2 must have the same free variables of sort $\mathcal{D}$.
3. If $\psi = \psi_1 \wedge \ldots \wedge \psi_n$ is a maximal formula of ϕ then all free variables of ψ of sort $\mathcal{D}$ must be limited in the following sense:

- A variable is limited if it is free in some g_i which is not negated.
- If ψ_i is $x = a$ or $a = x$ where $a \in Dom$ then x is limited.
- If ψ_i is $x = y$ or $y = x$ and y is limited then x is limited.

4. The negation operator $\neg$ may only apply to a subformula in a conjunction of the type discussed in the previous rule.

The above notion of safety takes into account *only* the data part of a predicate or a formula. The temporal part does *not* present problems thanks to the local constraints of tuples. For example, notice that the expression $f = \{R(B), t/\mathcal{P} : \neg P(t)\}$ is safe. If P is the relation $\{(A : t_1, CON : t_1 \leq 5)\}$ then evaluating f on P returns the *finite* generalized relation $R = \{(B : t, CON : t > 5)\}$. However, the expression $g = \{R(A, B), x/\mathcal{D}, t/\mathcal{P} : \neg Q(x, t)\}$ is not safe. This expression exhibits the usual problems of unsafe formulas with respect to its data domain.

[33] avoid dealing with safety issues by treating the data part of a tuple as a conjunction of equality constraints. We do not follow their approach because we believe that constraints should be reserved for the parts of the model that really gain from this kind of representation.

The following theorem demonstrates that safe expressions of relational calculus with temporal constraints can be evaluated bottom-up in closed form over generalized relational databases.

Theorem 8.1 *For every safe expression f over $\widetilde{S}$ in relational calculus with temporal constraints, there exists a generalized relational algebra expression e over $\widetilde{S}$ such that the following property holds. If $\widetilde{R}$ is a generalized relational database over $\widetilde{S}$ then $f(\widetilde{R}) = points(e(\widetilde{R}))$.*

Proof: The proof of this theorem is straightforward for somebody who is familiar with similar proofs from relational database theory [44, 63]. We will only give the generalized relational algebra expression for negated database predicates.

Let R be a generalized relation over scheme S. The generalized relational algebra expression for $\neg R(a_1, \ldots, a_k)$ is

$$(E^{|S_d|} \bowtie \{(B_1 : x_1, CON : true)\} \bowtie \cdots \bowtie \{(B_{|S_t|} : x_{|S_t|}, CON : true)\}) - e_R$$

where e_R is the generalized relational algebra expression for R, $B_1, \ldots, B_{|S_t|}$ are the attributes in S_t, $x_1, \ldots, x_{|S_t|}$ are u-variables of appropriate sorts, and E is an expression for the extended active data domain relative to the given calculus expression. E can be easily expressed in generalized relational algebra. ∎

The following theorem demonstrates that safe expressions of relational calculus with temporal constraints can be evaluated bottom-up in closed form over table databases. First, we present a technical lemma.

Lemma 8.1 *Let e be a pure table algebra expression over $\widetilde{S}$ and e' be its corresponding generalized relational algebra expression. Then $sem(e(\widetilde{T})) = \{points(e'(\widetilde{R}')) : \widetilde{R}' \in rep(\widetilde{T})\}$ for all table databases $\widetilde{T}$ over $\widetilde{S}$.*

Proof: This has already been shown explicitly in the proof of theorem 7.2. ∎

Theorem 8.2 *For every safe expression f over $\widetilde{S}$ in relational calculus with temporal constraints, there is a pure table algebra expression e over $\widetilde{S}$ such that the following property holds. If $\widetilde{T}$ is a table database over $\widetilde{S}$ then $f(\widetilde{T}) = sem(e(\widetilde{T}))$.*

Proof: Let f' be the generalized relational calculus expression corresponding to f. If e' is the generalized relational algebra expression provided by theorem 8.1 then e is the pure table algebra expression corresponding to e'. The result holds because of the following equalities:

$$f(\widetilde{T}) = \{f'(\widetilde{R}) : \ \widetilde{R} \in rep(\widetilde{T})\} = \{points(e'(\widetilde{R})) : \ \widetilde{R} \in rep(\widetilde{T})\} = sem(e(\widetilde{T}))$$

The last equality follows from lemma 8.1. ■

Finally, we demonstrate that safe expressions in modal relational calculus with temporal constraints can be evaluated bottom-up in closed form over table databases.

Theorem 8.3 *For every safe $\Box$-expression (resp. $\Diamond$-expression) f over $\widetilde{S}$ in modal relational calculus with temporal constraints, there is a table algebra $CERT$-expression (resp. $POSS$-expression) e over $\widetilde{S}$ such that the following property holds. If $\widetilde{T}$ is a table database over $\widetilde{S}$ then $f(\widetilde{T}) = sem(e(\widetilde{T}))$.*

Proof: Let f' be the safe expression of relational calculus with temporal constraints which corresponds to $\Box$-expression f. From lemma 5.1 we have $f(\widetilde{T}) = \bigcap f'(\widetilde{T})$. Then theorem 8.2 provides a pure table algebra expression e' such that $f(\widetilde{T}) = \bigcap sem(e(\widetilde{T}))$. Finally, lemma 6.1 gives $f(\widetilde{T}) = sem(CERT(e'(\widetilde{T})))$ which proves the theorem for $\Box$-expressions. The proof for $\Diamond$-expressions is similar. ■

9 Related Work

Let us now compare our proposal with some of the existing temporal database models. Although we used the rather generic term *temporal*, our model must be classified as *historical* in the taxonomy of [58]. Our model does not have any clear analogue to the notions of lifespan of a tuple or attribute as they appear in more traditional historical models. In our model temporal information is represented by special attributes whose values can be constants or variables of sort $\mathcal{P}$ or $\mathcal{I}$. This convention together with the constraint machinery results in a flexible and powerful representation scheme.

We could have avoided using u-variables and constraints for the representation of facts like "The vehicle is on service throughout the interval $\langle 9, 15 \rangle$" if we considered *two different kinds* of tables and modified the semantics of query evaluation.[12] The first kind of table would have been appropriate only for representing *facts* about the world. Time attributes of such tables would correspond to *valid time* in traditional historical models [58]. The second kind of table would have been appropriate only for *events* (instantaneous or not). Since we could achieve the desired functionality simply by using constraints, we decided not to adopt this option.

The database model which is most closely related to ours is [31]. Our model can be understood as an attempt to introduce indefinite temporal information in a formalism similar to the one proposed in [31]. [31] proposed a *generalized relational model* for representing temporal data. The distiguishing feature of this approach is that finite and infinite (but not indefinite) temporal data can be represented in a very natural way. The basic construct of this model is the linear repeating point. Given $c, p \in \mathbb{Z}$, a *linear repeating point* is a set $\{c + pn | \ n \in \mathbb{Z}\}$. If T is a set of linear repeating points and D a set of non-temporal data values then a *generalized tuple* is an element of $D^k \times T^k$ possibly constrained by a conjunction of inequalities of the form

$$X_i \leq X_j + a, \ X_i = X_j + a, \ X_i \leq a, \ X_i \geq a \text{ or } X_i = a$$

[12] This approach has been taken in [61].

where $1 \leq i,\ j \leq k$ and $a \in \mathbb{Z}$. A *generalized relation* is a finite set of generalized tuples of the same schema. [31] define *generalized relational algebra* operations and show that they can be computed with polynomial time *data complexity*. However, a significant problem lurks in the background. The database has to be "normalized" before the operation of projection can be computed. This normalization process can increase the size of a relation considerably. [50] discuss some practical alternatives for implementing generalized relational databases.

Another area where constraints play an active role is constraint logic programming. *Constraint logic programming* is an extension of logic programming with the goal of performing computation over an arbitrary *domain* (not necessarily the Herbrand universe) and replacing unification with the more powerful operation of *constraint solving* over this domain. Under this paradigm many constraint logic programming languages have been developed over the last few years (CLP($\mathcal{R}$) [29], Prolog III [14], CHIP [68] and others). Recently there have been attempts to apply the same ideas in the context of relational and deductive databases [33, 31, 52, 59, 28, 36]. A very general discussion of the issues involved in this exercise appears in [33]. Kanellakis, Kuper and Revesz proposed *constraint databases* as a powerful extension of relational databases. In their scheme a *constraint tuple* is a quantifier-free conjunction of constraints over a domain D. A relation is a finite set of tuples i.e., a disjunction of conjunctions of constraints. The query language for a constraint database is relational calculus with constraints over D. [33] studied constraint databases with several instances of a constraint language: polynomial inequality constraints over the reals, dense linear order constraints, equality constraints over an infinite domain and boolean equality constraints. In all cases they showed that queries in relational calculus with constraints can be evaluated in closed form with "low" computational complexity (except for boolean equality constraints). These results can be understood as theoretical arguments suggesting that constraint databases can become a practical technology.

Let us now discuss several data models which allow the representation of indefinite temporal information. [9] has studied databases with relative temporal information. The novel feature of [9] is that temporal relationships (e.g., before, after) can appear as attribute values in tuples (e.g., $\langle eventA, before, eventB\rangle$, $\langle eventA, after, 23Jan1992\rangle$ are tuples). This allows one to ask queries of the form "What is the temporal relationship between event A and event B ?". The model of [9] is in some sense weaker than ours since it can capture a less expressive class of temporal constraints (e.g., it cannot represent bounds on the distance of two time points; it cannot represent disjunctive constraints). In addition, its query language is not directly comparable with ours.

The recent model of [22] deals with indefinite temporal information. However, only two kinds of indefinite information are allowed. The first is a limited case of the kind of indefiniteness which can be expressed in our model: we can say that a tuple is in a relation at least over the interval $[a, b]$ and at most over the interval $[c, d]$ where $[a, b] \subseteq [c, d]$ (i.e., we are uncertain about the existence of the tuple during interval $[c, d] - [a, b]$). The second kind of indefinite information allows attribute values to be unknown during the period of (possible or certain) existence of the tuple they belong. This kind of indefinite information is not expressible in our model. However, tables can easily be extended to allow for constraints on the data part of a tuple. Then objects whose data attributes are infinite or indefinite can be modeled as well [33, 23, 27]. In this case, we can also guarantee that *every* query expressible in table calculus can be evaluated bottom up in closed form (this can be shown as in 8 or using the techniques of [33]).

The possible chronons data model [19] extends the temporal database model of [57] by allowing *indeterminate* events and intervals. An indeterminate event is one whose time of occurence cannot be precisely determined. In [19]'s terminology, an indeterminate event is a set of possible chronons together with a probability mass function which provides the probability that the event occured during each chronon. Indeterminate intervals are defined similarly. The user can query this information using an extension of the query language TQuel (an temporal extension to Quel developed in [57]). Striving for efficiency, the possible chronons model lacks the expressive power of our proposal. However, its probabilistic features should make it very attractive for some applications.

From the artificial intelligence literature, our work is most closely related to Telos [49], TSOS [45] and TMM [16]. Telos is a knowledge representation language based on a structurally object oriented model [49]. It allows the representation of indefinite temporal information using a tractable subset of the interval-to-interval qualitative relations introduced by Allen [1]. Telos cannot express duration constraints, does not allow modalities in the queries and cannot return constraints as answers to a query.

TSOS is a logic-based temporal reasoning system [45]. The most important features of TSOS is the ability to represent indefinite temporal information and to reason about different levels of time granularity. The query language of TSOS allows the expression of modal queries. This approach is different than ours because it is based on built-in specialized predicates rather than the integration of a first-order query language with modal operators. TSOS cannot return constraints as answers to a query.

TMM is probably the most powerful temporal reasoning system developed by artificial intelligence researchers [16, 15, 54]. Its capabilities extend from the representation of indefinite temporal information to reasoning about persistence and causality. The temporal constraint language used by TMM has the same expressiveness with the one used by our model. Finally, TMM allows the expression of modal queries but it cannot return constraints as answers to a query.

10 Conclusions

In this paper we showed how ideas from conditional tables [26], constraint databases [33, 31] and reasoning about temporal constraints [1, 17] can be integrated to provide a unified treatment of definite, indefinite, finite and infinite temporal data. There are several novel applications (e.g., planning, scheduling, project management, medical information systems, geographical information systems and natural language processing systems) where there is a strong practical need for temporal database models like the ones we proposed.

In [37, 35] we have also studied the complexity of query answering in generalized relational databases and temporal tables. Our most important result is that the worst-case complexity of query evaluation does *not* change when we go from relational databases to generalized relational databases. The same is true for relational databases with indefinite information vs. temporal tables. The next step will be to address implementation issues. This will be particularly challenging for the model of temporal tables due to the presence of indefinite information. Implementations of similar temporal reasoning systems [15], implementations of constraint logic programming languages [29], and implementations of constraint query languages [60, 59] will be the starting points of this investigation.

Our models should also be extended with periodic data. Given the work of [31], this should not be a difficult task but any implementation will be substantially more complicated. Finally, the models of this paper must be enhanced with deductive rules. Among other things this will allow us to express causality, persistence [16, 38], recursive queries and so on.

Acknowledgements

This research is part of the author's Ph.D. thesis. It was initiated at the University of Toronto and was continued at the Institute of Computer Science, FORTH and the National Technical

University of Athens. I would like to thank Timos Sellis, John Mylopoulos, Alberto Mendelzon, Yannis Vassiliou, Costas Courcoubetis, Dimitris Plexousakis and Thodoros Topaloglou for their encouragement and support. Financial support was gratefully received from the National Technical University of Athens, the Dept. of Computer Science, University of Toronto and the Institute of Computer Science, FORTH. Gosta Grahne deserves special thanks for sending me a copy of his thesis. Finally, I would like to thank one of the reviewers for his (or her) detailed and constructive comments.

References

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

[2] J.F. Allen. Towards a General Model of Action and Time. *Artificial Intelligence*, 23(2):123–154, July 1984.

[3] J.F. Allen, H. Kautz, and R. Pelavin, editors. *Reasoning About Plans*. Morgan-Kaufmann, 1991.

[4] F. Barbic and B. Pernici. Time Modeling in Office Information Systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 51–62, 1985.

[5] R. Barrera, A. Frank, and K. Al-Taha. Temporal Relations in GIS: A Workshop at the University of Maine. *Sigmod Record*, 20(3):85–91, 1991.

[6] Marianne Baudinet, Jan Chomicki, and Pierre Wolper. Temporal Deductive Databases. In Tansel, A. et. al., editor, *Temporal Databases: Theory, Design and Implementation*, chapter 13. Benjamin/Cummings Pub. Co., 1993.

[7] Marianne Baudinet, Marc Niezette, and Pierre Wolper. On the Representation of Infinite Temporal Data and Queries. In *Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 280–290, 1991.

[8] C. Brzoska. Temporal Logic Programming with Bounded Universal Goals. In *Proceedings of the 10th International Conference on Logic Programming*, pages 239–256, 1993.

[9] S. Chaudhuri. Temporal Relationships in Databases. In *Proceedings of the 14th VLDB Conference*, pages 160–170, 1988.

[10] J. Chomicki. Polynomial Time Query Processing in Temporal Deductive Databases. In *Proceedings of ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pages 379–391, 1990.

[11] J. Chomicki and T. Imielinski. Temporal Deductive Databases and Infinite Objects. In *Proceedings of ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pages 61–73, 1988.

[12] J. Chomicki and T. Imielinski. Relational Specifications of Infinite Query Answers. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 174–183, 1989.

[13] Jan Chomicki. Temporal Databases. Unpublished notes from a tutorial presented at the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 1993.

[14] A. Colmerauer. An Introduction to Prolog III. *Communications of ACM*, 33(7):69–90, July 1990.

[15] T. Dean. Using Temporal Hierarchies to Efficiently Maintain Large Temporal Databases. *Journal of ACM*, 36(4):687–718, 1989.

[16] T. Dean and D.V. McDermott. Temporal Data Base Management. *Artificial Intelligence*, 32:1–55, 1987.

[17] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49(1-3):61–95, 1991. Special Volume on Knowledge Representation.

[18] S. Dutta. Temporal Reasoning in Medical Expert Systems. In *Proceedings of IEEE Symposium on the Engineering of Computer-Based Medical Systems*, pages 118–122, 1988.

[19] C. Dyreson and R. Snodgrass. Valid-time Indeterminacy. In *Proceedings of the 9th International Conference on Data Engineering*, pages 335–343, 1993.

[20] H.B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972.

[21] M. Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling.* London: Pitman, 1987.

[22] S.K. Gadia, S. S. Nair, and Y.-C. Poon. Incomplete information in relation temporal databases. In *Proceedings of VLDB-92*, pages 395–406, 1992.

[23] Gosta Grahne. The Problem of Incomplete Information in Relational Databases. Technical Report Report A-1989-1, Department of Computer Science, University of Helsinki, Finland, 1989. Also published as Lecture Notes in Computer Science 554, Springer Verlag, 1991.

[24] G.E. Hughes and M.J. Cresswell. *An Introduction to Modal Logic.* Methuen, 1968.

[25] T. Imielinski. Incomplete Information in Logical Databases. *Data Engineering*, 12(2):29–39, 1989.

[26] T. Imielinski and W. Lipski. Incomplete Information in Relational Databases. *Journal of ACM*, 31(4):761–791, 1984.

[27] T. Imielinski, S. Naqvi, and K. Vadaparty. Incomplete Objects - A Data Model for Design and Planning Applications. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 288–297, 1991.

[28] Jaffar J., A. Brodsky, and M. Maher. Towards Practical Constraint Databases. In *Proceedings of 19th International Conference on Very Large Databases (VLDB-93)*, pages 567–580, 1993.

[29] J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. The CLP($\mathcal{R}$) language and system. *ACM Transaction on Programming Languages and Systems*, 14(3):339–395, July 1992.

[30] Joxan Jaffar and Jean-Louis Lassez. Constraint Logic Programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, pages 111–119, 1987.

[31] F. Kabanza, J.-M. Stevenne, and P. Wolper. Handling Infinite Temporal Data. In *Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 392–403, 1990.

[32] Paris Kanellakis. Elements of Relational Database Theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 17. North-Holland, 1990.

[33] Paris C. Kanellakis, Gabriel M. Kuper, and Peter Z. Revesz. Constraint Query Languages. In *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 299–313, 1990. Long version to appear in Journal of Computer and System Sciences.

[34] H. Kautz and P. Ladkin. Integrating Metric and Qualitative Temporal Reasoning. In *Proceedings of AAAI-91*, pages 241–246, 1991.

[35] M. Koubarakis. *Foundations of Temporal Constraint Databases.* PhD thesis, Computer Science Division, Dept. of Electrical and Computer Engineering, National Technical University of Athens, February 1994.

[36] Manolis Koubarakis. Representation and Querying in Temporal Databases: the Power of Temporal Constraints. In *Proceedings of the 9th International Conference on Data Engineering*, pages 327–334, April 1993.

[37] Manolis Koubarakis. Complexity Results for First-Order Theories of Temporal Constraints. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR'94)*, May 1994. Forthcoming.

[38] R.A Kowalski and M.J. Sergot. A Logic-based Calculus of Events. *New Generation Computing*, 1(4):67–95, 1986.

[39] Peter Ladkin. Primitives and Units for Time Specification. In *Proceedings of AAAI-86*, pages 354–359, 1986.

[40] Peter Ladkin. The Completeness of a Natural System for Reasoning with Time Intervals. In *Proceedings of IJCAI-87*, pages 462–467, 1987.

[41] Peter Ladkin. Satisfying First-Order Constraints About Time Intervals. In *Proceedings of AAAI-88*, pages 512–517, 1988.

[42] H.J. Levesque. Foundations of a Functional Approach to Knowledge Representation. *Artificial Intelligence*, 23:155–212, 1984.

[43] Witold Jr. Lipski. On Semantic Issues Connected with Incomplete Information Databases. *ACM Transcactions on Database Systems*, 4(3):262–296, September 1979.

[44] David Maier. *The theory of relational databases.* Computer Science Press, 1983.

[45] R. Maiocchi, B. Pernici, and F. Barbic. Automatic Deduction of Temporal Information. *ACM Transactions on Database Systems*, 17(4):647–688, 1992.

[46] D.V. McDermott. A Temporal Logic for Reasoning About Processes and Plans. *Cognitive Science*, 6:101–155, 1982.

[47] I. Meiri. Combining Qualitative and Quantitative Constraints in Temporal Reasoning. In *Proceedings of AAAI-91*, pages 260–267, 1991.

[48] S. Miller and L. Schubert. Using Specialists to Accelerate General Reasoning. In *Proceedings of AAAI-88*, pages 161–165, 1988.

[49] John Mylopoulos, Alex Borgida, Matthias Jarke, and Manolis Koubarakis. Telos: A Language for Representing Knowledge About Information Systems. *ACM Transactions on Information Systems*, 8(4):325–362, October 1990.

[50] M. Niezette and J.-M. Stevenne. An Efficient Symbolic Representation of Periodic Time. In *Proceedings of the 1st International Conference on Information and Knowledge Management*, 1992.

[51] Ray Reiter. On Integrity Constraints. In *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 97–111, Asilomar, CA, 1988.

[52] Peter Z. Revesz. A Closed Form for Datalog Queries with Integer Order. In *Proceedings of the 3rd International Conference on Database Theory*, pages 187–201, 1990. Long version to appear in Theoretical Computer Science.

[53] A. Sathi, M. Fox, and M. Greenberg. Representation of Activity Knowledge for Project Management. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(5):531–552, 1985.

[54] R. Schrag, M. Boddy, and J. Carciofini. Managing Disjunction for Practical Temporal Reasoning. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR'92)*, pages 36–46. Morgan Kaufmann, San Mateo, CA, October 1992.

[55] A. Schrijver, editor. *Theory of Integer and Linear Programming.* Wiley, 1986.

[56] Yoav Shoham. *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence.* MIT Press, 1987.

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

[58] R. Snodgrass. Temporal Databases. In *Proceedings of the International Conference on GIS – From Space to Territory: Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, pages 22–64, Pisa, Italy, September 1992.

[59] Divesh Srivastava. Subsumption and Indexing in Constraint Query Languages with Linear Arithmetic Constraints. In *Proceedings of the 2nd International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida, January 1992.

[60] Divesh Srivastava and Raghu Ramakrishnan. Pushing Constraint Selections. In *Proceedings of the 11th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 301–315, 1992.

[61] J.-M. Stevenne. A Model-Checking Approach to Temporal Reasoning. Presented at the 2nd Bar-Ilan Symposium on Foundations of Artificial Intelligence, January 1991.

[62] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and Snodgrass R., editors. *Temporal Databases: Theory, Design and Implementation.* Database Systems and Applications Series. Benjamin/Cummings Pub. Co., 1993.

[63] J. Ullman. *Principles of Data Base and Knowledge Base Systems*, volume 1. Computer Science Press, 1988.

[64] Peter van Beek. Reasoning About Qualitative Temporal Information. In *Proceedings of AAAI-90*, pages 728–734, 1990.

[65] Peter van Beek. Reasoning About Qualitative Temporal Information. *Artificial Intelligence*, 58:297–326, 1992.

[66] Peter van Beek and Robin Cohen. Exact and Approximate Reasoning about Temporal Relations. *Computational Intelligence*, 6:132–144, 1990.

[67] A. van Gelder and R.W. Topor. Safety and Translation of Relational Calculus Queries. *ACM Transactions on Database Systems*, 16(2):235–278, June 1991.

[68] P. van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[69] Marc Vilain and Henry Kautz. Constraint Propagation Algorithms for Temporal Reasoning. In *Proceedings of AAAI-86*, pages 377–382, 1986.

[70] Marc Vilain, Henry Kautz, and Peter van Beek. Constraint Propagation Algorithms for Temporal Reasoning: a Revised Report. In D.S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*, pages 373–381. Morgan Kaufmann, 1989.