**Artificial Intelligence Review manuscript No.**
(will be inserted by the editor)

# Expressiveness of Temporal Query Languages: On the Modelling of Intervals, Interval Relationships and States[⋆]

**Rodolfo Sabás Gómez · Juan Carlos Augusto**

Received: date / Revised version: date

**Abstract** Storing and retrieving time-related information are important, or even critical, tasks on many areas of Computer Science (CS) and in particular for Artificial Intelligence (AI). The expressive power of temporal databases/query languages has been studied from different perspectives, but the kind of temporal information they are able to store and retrieve is not always conveniently addressed. Here we assess a number of temporal query languages with respect to the modelling of time intervals, interval relationships and states, which can be thought of as the building blocks to represent and reason about a large and important class of historic information. To survey the facilities and issues which are particular to certain temporal query languages not only gives an idea about how useful they can be in particular contexts, but also gives an interesting insight in how these issues are, in many cases, ultimately inherent to the database paradigm.

While in the area of AI declarative languages are usually the preferred choice, other areas of CS heavily rely on the extended relational paradigm. This paper, then, will be concerned with the representation of historic information in two well known temporal query languages: *Templog* in the context of temporal deductive databases, and *TSQL2* in the context of temporal relational databases. We hope the results highlighted here will increase cross-fertilisation between different communities. This article can be related to recent publications drawing the attention towards the different approaches followed by the Databases and AI communities when using time-related concepts.

**Keywords** Temporal Deductive Databases · Temporal Relational Databases · Knowledge Representation · Temporal Logic

[⋆] Final version: R.S.Gomez and J.C.Augusto, “Expressiveness of Temporal Query Languages: On the modelling of intervals, interval relationships and states”. In: Artificial Intelligence Review, Springer Netherlands, ISSN 0269-2821 (Print) 1573-7462 (Online), DOI 10.1007/s10462-007-9051-4 (`http://www.springerlink.com/content/r3347p646152351m`).

Computing Laboratory, University of Kent, CT2 7NF Canterbury, Kent, UK E-mail: R.S.Gomez@kent.ac.uk · School of Computing and Mathematics, University of Ulster at Jordanstown, BT37 0QB Newtownabbey, Co. Antrim, UK E-mail: jc.augusto@ulster.ac.uk

## 1 Introduction

Temporal databases has recently become an active research area in computer science (Tansel et al 1993; Etzioni et al 1998; Morris and Khatib 1999; Goodwin and Trudel 2000; Bettini and Montanari 2001; Artale et al 2002; Reynolds and Sattar 2003)). This kind of databases offers the possibility to associate time to data and to deal with it in a way that non-temporal systems cannot handle, or in a much more convenient way. For example, in a medical database it is useful to store information related to the different stages a patient's health goes through. A banking system requires to store the time each transaction is done as well as expiration dates for its loans. A public transportation system must keep track of departing and arriving times for each unit. These are just a few from a plethora of contexts where dealing with time is fundamental for the success of the system.

There exist multiple levels to consider temporal issues in databases (Snodgrass and Ahn 1986). For example, database manager systems (DBMS) have traditionally offered some support for what is termed *user time*; this is usually represented with a special data type (e.g. `date`) but it is handled just as any other non-temporal attribute. For example, a company database may store the period in which every employee has held a particular position, by using a tuple with attributes such as the employee id, the name of the position and two attributes denoting the start and end dates of the period that position was held. However, in a conventional DBMS there are no primitives to deal with this information in such a way temporal consistency is preserved (e.g. tuple update or removal), neither are there primitives to conveniently perform complex forms of temporal reasoning. *Valid time* databases consider associate each tuple a validity period for that information (informally, the period in which the information is regarded as valid in the "real world"). *Transaction time* databases keep track of the time when information arrives to the database manager (also when it is deleted). Bitemporal databases support both valid and transaction time. In this paper, we will be concerned only with valid time.

The expressiveness of temporal database models and query languages has been studied from different perspectives. For instance, (Bohlen et al 1996) discusses a relationship between *TSQL2* and Temporal Logic, (Toman and Niwinski 1996) describes the class of first order queries that cannot be expressed in Temporal Logic, (Toman 1996) compares point-based vs. interval-based query languages, and (Baudinet et al 1993) surveys some languages regarding infinite temporal extensions. Other works addressing expressiveness issues include (McKenzie and Snodgrass 1989; Tansel and Tin 1998; Cobo and Augusto 1999). However, we believe that some important issues remain overlooked. The formal characterisation of a query language's expressive power usually receives the focus of attention, but the kind of temporal information that a query language is well suited to model and retrieve not always can be inferred directly from its expressive power, and language pragmatics often results poorly surveyed. For instance, knowing that a pair of languages are as expressive as a particular temporal logic does not always suffice to answer questions like *What kind of problems each language is best suited to deal with?*, *Are both languages prepared to handle the same problems?* or *Is similar information as easy to model and retrieve in one language as it is in the other?*. It is worth mentioning that query languages are not considered in isolation but related to a data model through their data manipulation primitives.

This paper addresses the facilities provided by temporal query languages for modelling time intervals, Allen's interval relationships and states (Allen 1983, 1984; Dowty 1986). These concepts are deeply related and are of paramount importance in valid time databases. Firstly, states stand for a possible way in which facts can be assigned temporal semantics. Briefly, facts are regarded as valid during time intervals according to point-based semantics (Jensen and Snodgrass 1996), which happens to be the kind of information that is usually required to be stored in valid time databases. Therefore the closer the query language expressions resemble states, the more natural it is to modelling temporal information. Finally, Allen's interval relationships can be thought of as powerful retrieval tools as they describe all possible relative locations (in time) between two intervals, and hence how a pair of facts may be located on the time axis. In addition, these relationships naturally arise in a wide range of application environments.

This paper, then, will be concerned with the representation of intervals, interval relationships and states in two well known temporal query languages: *Templog* (Abadi and Manna 1989) in the context of temporal deductive databases, and *TSQL2* (Snodgrass 1995) in the context of temporal relational databases. We will also contrast *Templog* with *Datalog*$_{1S}$ (Chomicki 1990a), and *TSQL2* with *TQuel* (Snodgrass 1987). A comparison between these languages serves to complement existing surveys, and helps in realising how subtle differences in expressive power, data representation models or even the choice of predefined temporal operators impact on the modelling of intervals, interval relationships and states.

From a wider perspective this work adds to previous contributions (Gómez and Augusto 2000; Galton and Augusto 2002; Gómez and Augusto 2004) rasing awareness in the Databases and AI communities about the potential benefits that considering the mutual approaches may bring to each other.

The paper is organised as follows. Section 2 describes a formalisation of time intervals, interval relationships and states; and discusses the importance of these concepts on revealing the value of temporal query languages from a user's perspective. At this respect, *Templog* and *TSQL2* are analysed in sections 3 and 4, respectively. Among other issues, a comparison between *Templog* and *Datalog*$_{1S}$ is offered in section 3.5. Similarly, *TSQL2* and *TQuel* are compared in section 4.5. Conclusions are given in section 5.

## 2 Interval, Interval Relationships and States

This section defines intervals, interval relationships and states in their most usual interpretation, and their relevance in representing valid time information. In accordance with the consensus glossaries presented in (Jensen et al 1998; Bettini et al 1998a), we define a *time domain* as a couple $\langle \mathcal{T}, \leq \rangle$ where $\mathcal{T}$ is a non-empty set of *time instants* and $\leq$ is a total order on $\mathcal{T}$. Bounded and unbounded discrete-time models can be defined over this domain. For instance, in models with initial time there exists a distinctive instant $o \in \mathcal{T}$ such that $o \leq i$ for all $i \in \mathcal{T}$.

**Definition 1** A time interval is a set of consecutive instants. A closed interval $I$ with bounding instants $i^-, i^+ \in \mathcal{T}$, $i^- \leq i^+$, is defined as $I = [i^-, i^+] = \{i \in \mathcal{T} \mid i^- \leq i \leq i^+\}$.

Intervals can be thought as being one of the building blocks of valid time information. Query languages that are well suited to handle intervals provide a compact representation of temporal validity, i.e., an efficient way of assigning temporal semantics to facts. Later in this section we will see their relevance on defining states.

*Example 1* Intervals are handled in *TSQL2* (see details in section 4) through the `PERIOD` predefined data type. For instance, `PERIOD '[2002, 2003]'`.

Note, also, that our time domain is general enough to support *chronons* and *granules* (Jensen et al 1998; Bettini et al 1998a). A chronon is defined as a non-decomposable time interval of some fixed, minimum duration (which is typically set by applications). Data models related to some query languages represent the time line by a sequence of chronons, and granularities, e.g., days or years, are built by indexing sets of consecutive chronons.

*Interval relationships* were proposed by Hamblin (Hamblin 1972) and later explored by Allen (Allen 1983, 1984) in the context of temporal reasoning, where intervals are the temporal primitives and facts can be assigned to them with such different meanings as properties, processes and events. They can be defined over $\mathcal{T}$ as follows:

**Definition 2** An *interval relationship* is a predicate over $\mathcal{I} \times \mathcal{I}$, where $\mathcal{I}$ is the set of all closed intervals over $\mathcal{T}$. Let $I = [i^-, i^+]$ and $J = [j^-, j^+]$ be two intervals, then interval relationships are interpreted as follows (inverse relationships can be modelled by swapping predicate arguments):

$$\begin{array}{lll}
\mathcal{T} \models \mathit{before}(I, J) & \text{iff} & \mathcal{T} \models i^+ < j^- \\
\mathcal{T} \models \mathit{meets}(I, J) & \text{iff} & \mathcal{T} \models i^+ = j^- \\
\mathcal{T} \models \mathit{overlaps}(I, J) & \text{iff} & \mathcal{T} \models i^- < j^- < i^+ < j^+ \\
\mathcal{T} \models \mathit{during}(I, J) & \text{iff} & \mathcal{T} \models j^- < i^- < i^+ < j^+ \\
\mathcal{T} \models \mathit{starts}(I, J) & \text{iff} & \mathcal{T} \models i^- = j^- < i^+ < j^+ \\
\mathcal{T} \models \mathit{finishes}(I, J) & \text{iff} & \mathcal{T} \models j^- < i^- < i^+ = j^+ \\
\mathcal{T} \models \mathit{equals}(I, J) & \text{iff} & \mathcal{T} \models i^- = j^- < i^+ = j^+
\end{array}$$

*Example 2* The *TSQL2* predefined function `PRECEDES`, which can be used in `WHERE`-clauses of `SELECT` statements (see details in section 4), is semantically equivalent to the interval relationship *before()*. For instance, the following expression is true:

```
PERIOD '[1999, 2000]' PRECEDES PERIOD '[2002, 2003]'
```

Interval relationships describe every possible way in which two intervals may be positioned on the time axis, and by extension between a pair of facts if they are assigned temporal validity over intervals. This completeness makes the relationship set a sound vehicle to compare how conveniently temporal query languages retrieve information.

*States* can be thought of as one of many possible ways in which information can be assigned temporal semantics. They have been studied by areas such as Philosophy, Linguistics and Artificial Intelligence. For instance, they are considered one of the classes in which human beings capture reality through language expressions, e.g., as *stative sentences* (Dowty 1986); or, from other perspective, as a way in which facts can be associated to time (Allen 1984; Galton 2005).

States may be regarded as statements which are considered true over time intervals, called *validity intervals*. For example, the sentence *John worked for the company from 1990 to 1998* denotes a state in which the fact *John works for the company* is considered true over the interval than ranges from 1990 to 1998. Moreover, states hold a distinctive property, usually known in the TDB an AI communities as *downward hereditary* (Allen 1984; Bettini et al 1998b); if a state holds over interval $I$, then it also holds over any subinterval of $I$. For example, that *John worked for the company from 1990 to 1998* implies that *John worked for the company from 1995 to 1997*. States can be expressed by temporal databases if facts are assigned intervals according to *point-based semantics* (Jensen and Snodgrass 1996; Bettini et al 1998b); a fact is true over a given interval if and only if it is true at every instant of that interval. Formally, states can be defined as follows:

**Definition 3** Let the pair $\langle \mathcal{D}, \mathcal{T} \rangle$ represent the structure of a given temporal query language $\mathcal{L}$, where $\mathcal{D}$ denotes the data model, i.e., a set of facts which are expressible by the language, and $\mathcal{T}$ its temporal structure. In addition, let $\mathcal{I}$ denote the set of all possible intervals over $\mathcal{T}$. We will say that *states* can be modelled in $\mathcal{L}$ if a mapping $S : \mathcal{D} \rightarrow 2^{\mathcal{I}}$ can be defined, such that for every pair $(d, \{I_1, \dots, I_n\}) \in S$ the fact $d$ is considered valid over every instant $i \in I_j$, for all $1 \leq j \leq n$.

*Example 3* The following tuple, extracted from a *TSQL2* valid time table (see details in section 4), can be thought of as modelling the state "Ann Smith worked for the company from 1990 to 1994, and then again from 1998 until 2002". *TSQL2* regards the information encoded by this tuple as valid during every year in $\{$1990, …, 1994, 1998, …, 2002$\}$.

| NAME | VALID TIME |
|---|---|
| Ann Smith | $\{$'[1990-1994]' $\cup$ '[1998-2002]'$\}$ |

Information in temporal databases are very often required to be stored as states. Query languages that can handle states are thus able to model a wide range of situations, which adds real value from the user's perspective. In what follows we will assess how some well known temporal query languages handle intervals, interval relationships and states. Let us note that this paper will not deal with issues such as the implication of open intervals in databases (Clifford et al 1997), relationships on open-intervals (Freksa 1992) or indeterminacy of information (Dyreson and Snodgrass 1998). While all of these aspects are certainly interesting, we believe their inclusion in this paper will make it exceed a reasonable length.

## 3 Intervals, Interval Relationships and States in Templog

This section is devoted to show how intervals, interval relationships and states are supported by *Templog*. This fact may seem surprising since Allen's relationships and states are build over intervals, and *Templog* does not provide them as a primitive concept. However, we will see that under certain modelling assumptions intervals can be implicitly represented if we relate them to the occurrence of certain context-dependent events. In addition, the representation of states in *Templog* is made possible as the language assigns validity to predicates according to point-based semantics. Therefore, we will see that Allen's relationships can be expressed by comparing the interval bounding events by means of temporal logic operators such as $\Diamond$; and that states can be represented as facts whose validity extends between two bounding events, by means of *Templog*'s inference rules and recursion.

## 3.1 Language overview

*Templog* (Abadi and Manna 1989; Baudinet 1989, 1992) is a syntactic extension of logic programming to linear-time temporal logic. Time is then isomorphic to $\mathbb{N}$, i.e., linear, discrete, with initial time and unbounded future. In this language, predicates may vary with time, but the time point they refer to is defined implicitly by temporal operators rather by an explicit temporal argument.

The only temporal operators used in *Templog* are $\bigcirc$ (*next*), which refers to the next time instant, $\Box$ (*always*), which refers to the present and all the future time instants, and $\Diamond$ (eventually), the dual of $\Box$, which refers to the present or to some future time instant.

The abstract syntax for *Templog* clauses is defined by the following grammar, where $A$ stands for an atom; $\varepsilon$ denotes an empty formula, "$\leftarrow$" the logical implication operator and a comma "," in a body the conjunction operator. $N$ stands for a *next-atom*, that is, we will use $\bigcirc^n A$ to denote $\underbrace{\bigcirc \ldots \bigcirc}_{n\ times} A$.

| | | |
|---|---|---|
| *Body:* | $B$ | $::= \varepsilon \vert A \vert B_1, B_2 \vert \bigcirc B \vert \Diamond B$ |
| *Initial Clause:* | $IC$ | $::= N \leftarrow B \vert \Box N \leftarrow B$ |
| *Permanent Clause:* | $PC$ | $::= \Box(N \leftarrow B)$ |
| *Program Clause:* | $O$ | $::= IC \vert PC$ |
| *Goal Clause:* | $G$ | $::= \leftarrow B$ |

*Initial clauses* describe statements that holds at the initial time; *permanent clauses* express statements that hold at any time instant. Program and goal clauses are assumed to be universally quantified, as in classical logic programming (Lloyd 1987). Each *Templog* program is a finite set of program clauses. Computation in *Templog* programs is based on a temporal logic resolution method, termed TSLD-resolution (Abadi and Manna 1989; Baudinet 1995). Semantics for temporal logic formulas are provided w.r.t. a temporal interpretation $D$ that is an infinite sequence $D_0, D_1, \ldots$ of classical first-order interpretations (one classical interpretation for each time instant). In *Templog*, only predicates symbols have time-varying meanings; constants and function symbols are assumed to be independent of time. *Templog* operators are interpreted as follows:

$\models_{D_i} \bigcirc F$ iff $\models_{D_{i+1}} F$
$\models_{D_i} \Box F$ iff for every $j \in \mathbb{N}$, $\models_{D_{i+j}} F$
$\models_{D_i} \Diamond F$ iff for some $j \in \mathbb{N}$, $\models_{D_{i+j}} F$

```
start_work(m1)
○ start_work(m2)
□( ○² stop_work(m1) ← start_work(m1))
□( ○⁴ stop_work(m2) ← start_work(m2))
□( ○² start_work(M) ← stop_work(M))
```

**Fig. 1** A simple *Templog* program.

A formula $F$ is satisfiable in a given interpretation $D$ iff $\models_{D_0} F$. A formula is valid if it is satisfiable in all possible interpretations.

*Templog* cannot (naturally) deal with contexts where the use of explicit time references, or database updates are the rule rather than the exception (Kowalski 1992). Because of its roots in temporal logic, *Templog* is best suited to deductive databases and, in general, applications where temporal reasoning and the concise representation of relative, possibly infinite information is required (e.g. periodic information). Thus, our elaboration on how intervals, Allen's relationships and states can be represented in *Templog* will take into account those contexts where the language would find a more natural application.

Figure 1 shows a *Templog* program where the alternating use of two machines, `m1` and `m2`, is represented. The program depicts a cycle where `m1` starts initially, `m2` starts 1 time units after that, `m1` works in periods of 2 time units (and `m2` in periods of 3 time units), and both machines idle for 2 time units between working periods.

## 3.2 Time intervals in *Templog*

Since explicit temporal references are not supported by *Templog*, we will assume that intervals will be related to states, i.e. a certain fact which is considered valid on a given period of time. Intervals, then, can be represented by a pair of predicates which denote the occurrence of those events which bound the corresponding state. Because the same bounding event may have multiple occurrences, these occurrences are also used to uniquely identify a given interval. Thus, intervals can be represented by a pair of predicates `begin(`$\bar{i}$`)` and `end(`$\bar{i}$`)`, where $\bar{i}$ represents a list of attributes which uniquely identify the interval in question, e.g. a state name, and a particular instance number which is related to a specific occurrence of bounding events (a full characterisation of states will be discussed later, in section 3.4). Notice that more convenient representations may exist. Since, in general, this will depend on the problem being modelled, we have proposed just a possible solution which might accommodate a number of commonly found scenarios.

Figure 2 shows how the program of Figure 1 can be modified to represent those intervals where each machine is operational. Such a state (`working`) is bounded by the events corresponding to a machine starting and stopping. Correspondingly, the predicates `start_work` and `stop_work` now feature an extra parameter (a natural number) identifying the event occurrence. Notice that the program offers, indeed, a concise representation of infinitely periodic intervals, e.g. that the machine `m1` is operational during $[4k, 4k+2], k \in \mathbb{N}$.

```
start_work(m1,1)
○ start_work(m2,1)
□( ○² stop_work(m1,N) ← start_work(m1,N))
□( ○⁴ stop_work(m2,N) ← start_work(m2,N))
□( ○² start_work(M,N+1) ← stop_work(M,N))

□(begin(working,M,N) ← start_work(M,N))
□(end(working,M,N) ← stop_work(M,N))
```

**Fig. 2** Representing time intervals in *Templog*.

## 3.3 Modelling interval relationships

Since Allen's relationships are defined in such a way that actual temporal spans between intervals are abstracted away, *Templog*'s modal operators provide a natural way of representing the relative position between two intervals by comparing the bounding events (see semantics of Allen's relationships in def. 2). Notice that we use "$\Diamond_\circ$" to denote "$\Diamond\bigcirc$", which in turn represents the relational operator "$<$" ($\Diamond$ is reflexive) between a pair of instants. Let $\bar{i}$, $\bar{j}$ be the list of attributes which uniquely identify intervals $I$ and $J$, respectively. Intervals relationships can be modelled by predicates `before`, `meets`, etc., as shown below. Without loss of generality, we assume that these predicates are time-independent.

```
□ before(ī,j̄)
          ← ◇(end(ī),◇∘begin(j̄))
□ meets(ī,j̄)
          ← ◇(end(ī),begin(j̄))
□ overlaps(ī,j̄)
          ← ◇(begin(ī),◇∘(begin(j̄),◇∘(end(ī),◇∘end(j̄))))
□ during(ī,j̄)
          ← ◇(begin(j̄),◇∘(begin(ī),◇∘(end(ī),◇∘end(j̄))))
□ starts(ī,j̄)
          ← ◇(begin(ī),begin(j̄),◇∘(end(ī),◇∘end(j̄)))
□ finishes(ī,j̄)
          ← ◇(begin(j̄),◇∘(begin(ī),◇∘(end(ī),end(j̄))))
□ equals(ī,j̄)
          ← ◇(begin(ī),begin(j̄),◇∘(end(ī),end(j̄)))
```

Notice that these predicates are just templates, which have to be adapted to particular contexts. For example, and following the example shown in Figure 2, suppose that we want to check whether it is possible that machine `m1` finishes its task before `m2` starts working on its own. This check could be done by asserting predicate `before/4`, as shown in Figure 3, and querying the goal

```
← before(m1,N,m2,N).
```

```
start_work(m1,1)
○ start_work(m2,1)
□( ○^2 stop_work(m1,N) ← start_work(m1,N))
□( ○^4 stop_work(m2,N) ← start_work(m2,N))
□( ○^2 start_work(M,N+1) ← stop_work(M,N))
□(begin(working,M,N) ← start_work(M,N))
□(end(working,M,N) ← stop_work(M,N))

□ before(M1,N1,M2,N2) ← ◇(end(working,M1,N1),
                          ◇_○begin(working,M2,N2))
```

**Fig. 3** Representing interval relationships in *Templog*.

```
start_work(m1,1)
○ start_work(m2,1)
□( ○^2 stop_work(m1,N) ← start_work(m1,N))
□( ○^4 stop_work(m2,N) ← start_work(m2,N))
□( ○^2 start_work(M,N+1) ← stop_work(M,N))
□(begin(working,M,N) ← start_work(M,N))
□(end(working,M,N) ← stop_work(M,N))

□ (valid(working,M,N) ← begin(working,M,N))
□ ( ○ valid(working,M,N) ← valid(working,M,N),
                             ◇_○ end(working,M,N))
□ (state(working,M) ← valid(working,M,N))
```

**Fig. 4** Representing states in *Templog*.

## 3.4 Modelling states

States can be modelled in *Templog* by a program where a) the state's validity intervals have been asserted, and b) a predicate denoting the state in question is made valid at every point included in a validity interval. We have shown, already, that validity intervals can be modelled by asserting a pair of predicates `begin`, `end` denoting the interval's bounding events. For example, Figure 4 shows how the state of a machine being operational (`working`) could be represented. Notice that the predicate `valid` is used to assert the validity of `state` at every point in time between `begin` and `end`.

## 3.5 Discussion

**Templog and Datalog$_{1S}$.** The expressiveness of the function-free subset of *Templog* is known to be equivalent to that of *Datalog*$_{1S}$ (Chomicki 1990a), a minimal extension of *Datalog* (Gallaire et al 1984; Grant and Minker 1992) (the subset of function-free Horn-clause logic programs) where predicates are allowed to contain one *temporal* argument where a successor function can be applied. Consequently, both *Templog* and *Datalog*$_{1S}$ have been proposed as suitable query languages for temporal deductive databases (Baudinet et al 1993). However, the limitation of *Datalog*$_{1S}$ to allow the successor function to be applied to at most one predicate argument severely limits

```
start_work(m1,0).
start_work(m2,1).
start_work(m3,0).
stop_work(m3,5).
stop_work(m1,T+2) :- start_work(m1,T).
stop_work(m2,T+4) :- start_work(m2,T).
start_work(M,T+2) :- stop_work(M,T).

begin(working,M,T) :- start_work(M,T).
end(working,M,T) :- stop_work(M,T).
```

**Fig. 5** The limits of *Datalog*$_{1S}$.

the modelling of intervals (and consequently, that of Allen's relationships and states). For example, Figure 5 shows a *Datalog*$_{1S}$ program similar to that of Figure 2, but extended with a third machine `m3` working just during the interval $[0, 5]$. Here, the last argument in every predicate is assumed to be the temporal parameter. Notice that predicates `begin/3` and `end/3` correctly represent the single working interval for `m3` ($[0, 5]$), but they cannot distinguish the working intervals for `m1` or `m2` because a pair [`begin`, `end`] does not necessarily correspond to a *matching* pair [`start_work`, `stop_work`]. For example, they represent both $[0, 2]$ and $[0, 6]$, although the proper intervals were meant to be $[0, 2]$, $[4, 6]$, $\ldots [4k, 4k + 2]$, $k \in \mathbb{N}$. Notice that $[0, 6]$ is represented as a consequence of pairing the first occurrence of `start_work` at time $0$ and the second occurrence of `stop_work` at time $6$. This problem is the result of *Datalog*$_{1S}$ not being expressive enough to distinguish between different occurrences of the same bounding event, which in *Templog* was made possible by adding an extra data parameter and a rule to increment it every time a new occurrence was identified (see Figure 2).

However, *Datalog*$_{1S}$ is expressive enough to deal with settings where states are not only assumed to be represented by a pair of bounding events, but also where multiple occurrences of the same event do not happen. If this is so, then the following queries can be expressed in *Datalog*$_{1S}$: a) whether the validity intervals corresponding to two different states satisfy a given Allen's relationship, b) whether a given state is valid at a particular point in time, and c) whether a given state is valid at a particular interval. All of these are recognition queries (i.e. they have yes/no answers). Generation queries are also possible (e.g. those which returns the set of states which hold simultaneously at a particular point in time), but generation queries with infinite answers require a more involved evaluation technique. This includes the generation of a finite model both for the Herbrand model of the program in question, and for the answer to the query (Baudinet et al 1993).

Figure 6 shows a *Datalog*$_{1S}$ program where the database is composed of 3 pairs of tuples [`start_work/2`, `stop_work/2`] denoting the working intervals $[0, 2]$, $[5, 9]$ and $[7, 11]$, for the machines `m1`, `m2` and `m3` respectively. The state of a machine being operational is represented by the predicate `working/2`. This predicate is assigned temporal validity by conjoining two auxiliary predicates `forward/2` and `backward/2`, to represent those time points where a machine starts or has started, and stopped or will stop (respectively). The remaining auxiliary predicates support the definition of `before/2` and `overlaps/2`, which in turn represent the corresponding Allen's relationships between the working intervals of two machines `X` and `Y`. For example, `before(X,Y)` holds if Allen's relationship *before(I,J)* holds, where *I* and *J* are the working intervals of machines `X` and `Y`, respectively. The difficulty in expressing Allen's relationships in *Datalog*$_{1S}$ (notice the definition of `overlaps/2`) comes from the fact that the relation $<$ between time points is not directly available in the language. Thus, predicate `started(M,T)` (respectively `stopped(M,T)`) holds at all time-points `T` after machine `M` started (stopped). Similarly, predicate `started_started(X,Y,T)` (`stopped_stopped(X,Y,T)`) holds at all `T` after both machines `X` and `Y` have started (stopped), provided `X` started (stopped) before `Y`; and predicate `started_stopped(X,Y,T)` denotes all time-points `T` after `Y` stopped, provided `X` started before.

With these auxiliary predicates, `overlaps(X,Y)` can be intuitively understood to hold if there exist a number of time points $t_1 < t_2 < t_3 < t_4$ such that `X` started at $t_1$, `Y` started at $t_2$, `X` stopped at $t_3$ and `Y` stopped at $t_4$. Indeed, if this is the case, then `T` $= t_4$ satisfies `started_started(X,Y,T)`, `started_stopped(X,Y,T)` and `stopped_stopped(X,Y,T)`. Figure 7 depicts the situation. Notice that the validity intervals for the state of a machine being operational are, for `X` and `Y`, $I = [t_1, t_3]$ and $J = [t_2, t_4]$ (respectively), and effectively `overlaps(X,Y)` holds if *overlaps*$(I, J)$ holds (compare with the semantics given for the Allen's relationship *overlaps*$(I, J)$ in section 2). Finally, the following queries are possible:

– Is the machine `m1` working at time 1?
  (Goal "`:- working(m1,1)`")
– Is the machine `m2` working during $[6, 8]$?
  (Goal "`:- working(m2,6), working(m2,8)`")
– Do working intervals for machines `m2` and `m3` overlaps?
  (Goal "`:- overlaps(m2,m3)`")

It is worth mentioning that other extensions of *Datalog* have been proposed in the literature, in which the modelling of intervals, Allen's relationships and states might become an easier task. These extensions include stratified negation (Baudinet et al 1993; Chomicki 1994), and integer and periodicity constraints (Revesz 1993; Toman et al 1994). However, these languages are still not expressive enough as to model states where bounding events can have multiple occurrences. Other extensions overcome this problem, such as adding integer constraints and stratified negation (Revesz 1993), or allowing predicates to have an arbitrary number of temporal parameters (where a successor function can be applied) (Baudinet et al 1991); but query evaluation for these languages is, in general, not guaranteed to terminate (they bring the expressive power of Turing-computable functions).

**Templog and the Event Calculus.** Readers would acknowledge that our approach for modelling intervals and states has much in common with the representation of time periods and relationships in the Event Calculus (Kowalski and Sergot 1986; Kowalski 1992). Some features of the Event Calculus, such as negation and explicit temporal references, make the representation of these temporal entities easier to express than it is in *Templog*. However in general this depends on the intended application, and *Templog* might be the preferred choice when infinite databases must be

```
% -- the database (no multiple occurrences for the same bounding event)

start_work(m1,0).
stop_work(m1,2).
start_work(m2,5).
stop_work(m2,9).
start_work(m3,7).
stop_work(m3,11).

% -- representing the "working" state

forward(M,T) :- start_work(M,T).
forward(M,T+1) :- forward(M,T).
backward(M,T) :- stop_work(M,T).
backward(M,T) :- backward(M,T+1).

working(M,T) :- forward(M,T), backward(M,T).

% -- representing Allen's relationships ("before" and "overlaps")

started(M,T+1) :- start_work(M,T).
started(M,T+1) :- started(M,T).

stopped(M,T+1) :- stop_work(M,T).
stopped(M,T+1) :- stopped(M,T).

started_started(X,Y,T) :- start_work(Y,T), started(X,T).
started_started(X,Y,T+1) :- started_started(X,Y,T).

started_stopped(X,Y,T) :- stop_work(Y,T), started(X,T).
started_stopped(X,Y,T+1) :- started_stopped(X,Y,T).

stopped_stopped(X,Y,T) :- stop_work(Y,T), stopped(X,T).
stopped_stopped(X,Y,T+1) :- stopped_stopped(X,Y,T).

before(X,Y) :- start_work(Y,T), stopped(X,T).
overlaps(X,Y) :- started_started(X,Y,T), started_stopped(Y,X,T),
                 stopped_stopped(X,Y,T).
```

**Fig. 6** Allen's relationships and states in *Datalog*$_{1S}$.

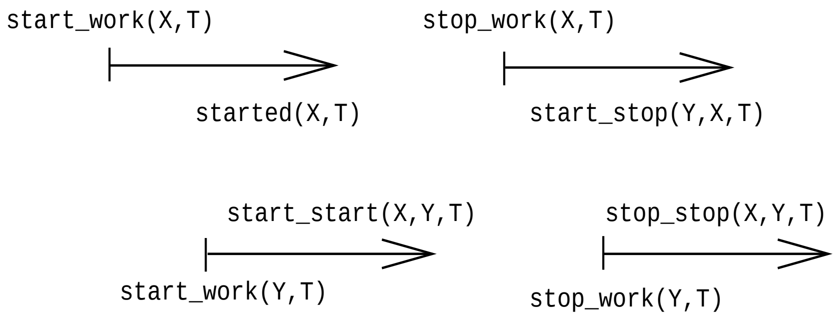

**Fig. 7** Intuitive interpretation of `overlaps(X,Y)`.

represented. We will not extend the comparison between *Templog* and the Event Calculus any further, but the reader will find extensive research done on both theory and applications of the Event Calculus in the literature (Chittaro and Montanari 1996; Cervesato et al 2000).

**Coalescing in Templog.** It turns out that because intervals have to be "programmed" ad-hoc in *Templog* by using the language primitives, maximal intervals cannot be feasibly enforced (usually known as *coalescing* (Tansel et al 1993)). We have seen that, since explicit time-references are not supported, intervals could be represented by predicates asserting the occurrence of bounding events. Then, coalescing would involve a revision of the database which, as discussed by (Kowalski 1992), may result in a number of update operations which is disproportionate w.r.t. the complexity of the coalescing itself. Also, it is not difficult to see that this problem is even worse for databases representing periodic information. For example, given the following two intervals:

```
begin(working,m1,1)
○² end(working,m1,1)
○³ begin(working,m1,2)
○⁵ end(working,m1,2)
```

we can see that, in this case, the state of *working* is related to $[0, 2] \cup [3, 5]$, instead of the maximal interval $[0, 5]$ (time in *Templog* is discrete). Thus, coalescing these two non-maximal intervals requires two predicate removals ($\bigcirc^2$ `end/3` and $\bigcirc^3$ `begin/3`) and one attribute update (the occurrence number in $\bigcirc^5$ `end/3` is set to `1` instead of `2`), giving:

```
begin(working,m1,1)
○⁵ end(working,m1,1)
```

Finally, let us mention that coalescing is not supported in $Datalog_{1S}$, either.

**Computability of Templog.** Baudinet (Baudinet 1989) showed that TSLD-resolution is both sound and complete for *Templog*. However, a number of results appear in (Baudinet et al 1993) which might help to have a better account of the implications of evaluating *Templog* programs. For one thing, it is suggested that TSLD-resolution would require a form of term-size check to guarantee termination in the presence of functional symbols. For the function-free subset of *Templog*, TSLD-resolution terminates for queries with finite answers. Also, it is shown how bottom-up evaluation (as for *Datalog*$_{1S}$) can be modified to deal with queries with infinite answers; but it is not clear how this could be applied for a top-down evaluation technique such as TSLD-resolution. Regarding the complexity of evaluating a given recognition query, the function-free subset of *Templog* is PSPACE-complete in terms of the size of the database (extrapolating results given for *Datalog*$_{1S}$). Although we are not aware of such results, we conjecture that tractable subsets might be found for *Templog* which can be evaluated in polynomial time (as done for *Datalog*$_{1S}$ in (Chomicki 1990b)).

## 4 Intervals, Interval Relationships and States in TSQL2

Intervals, interval relationships and states are all concepts which can be considered primitive in *TSLQ2* (Snodgrass 1995); thus, we will see that contexts in which these concepts play a central role, can be modelled quite naturally. The presentation of *TSLQ2* in this section will focus in the contrast between this language, which is representative of the relational database paradigm, and *Templog*/$Datalog_{1S}$ as deductive approaches. The general syntax of *TSLQ2* specifications, and further syntactic and behavioral details of *TSLQ2* operators presented in this section, can be found in (Snodgrass 1995).

### 4.1 Language overview

*TSQL2* is a temporal relational query language, based on the SQL-92 standard (Tansel et al 1993). It supports user-defined, valid and transaction times. The temporal structure is linear, discrete and bounded. Time references are stored in fixed-size structures called *timestamps*. The time line is partitioned into chronons, and several granularities and calendars, both predefined an user-defined, can be built over chronons with different grouping schemas. Timestamps are expressed as values with associated granularity and calendar. For example, the expression `DATE '18/06/2003'` is a timestamp denoting a specific day in the calendar (18 June, 2003).

### 4.2 Modelling time intervals

Intervals in *TSQL2* are supported via the data type `PERIOD`. Conceptually, intervals are sets of consecutive granules represented by a pair of timestamps in the same granularity (denoting the interval boundaries). For example, the expression `PERIOD '[1/2003-6/2003]'` denotes the set {*Jan/03*, *Feb/03*, . . ., *Jun/03*}.

Unlike in *Templog*, intervals are entities themselves, not necessarily attached to facts. *TSLQ2* provides a number of functions to manipulate intervals. Among other operations, `BEGIN` and `END` return the interval timestamps; and `INTERSECT` and `+` return, respectively, the intersection and union of two intervals. Notice that the the union of two intervals may result into a set of intervals. For example,

```
PERIOD '[3/2002-5/2002]' + PERIOD '[7/2002-12/2002]'
```

yields the set,

```
{ '[3/2002-5/2002]' ∪ '[7/2002-12/2002]' }
```

*TSQL2* can also handle sets of intervals as entities (called temporal *elements*), and provides the usual set operations (e.g. intersection, union and difference) and the functions `FIRST` and `LAST` to return the first and last interval of a set (in chronological order).

**Table 1** Interval relationships in *TSQL2*.

| Interval relationship | Equivalent *TSQL2* expression |
|---|---|
| *before(I,J)* | `I PRECEDES J` |
| *meets(I,J)* | `END(I) = BEGIN(J)` |
| *overlaps(I,J)* | `BEGIN(I) PRECEDES BEGIN(J) AND END(I) PRECEDES END(J)` |
| *during(I,J)* | `BEGIN(J) PRECEDES BEGIN(I) AND END(I) PRECEDES END(J)` |
| *starts(I,J)* | `BEGIN(I) = BEGIN(J) AND END(I) PRECEDES END(J)` |
| *finishes(I,J)* | `BEGIN(J) PRECEDES BEGIN(I) AND END(I) = END(J)` |
| *equals(I,J)* | `I = J` |

## 4.3 Modelling interval relationships

Interval relationships can be easily defined via the relational operators `=` (equals) and `PRECEDES`, which compares two timestamps, and the functions `BEGIN` and `END` to extract the timestamp from the interval in question. These definitions can be seen in Table 1 (Snodgrass 1995), where `I` and `J` denote two intervals.

## 4.4 Modelling states

States can be expressed as valid time tuples. *TSQL2* supports valid time tables (called *state tables*), where tuples are assigned a set of maximal, non adjacent intervals (valid time elements). Coalescing is automatically handled by the DBMS whenever valid time elements are updated.

The information expressed in the tuple attributes is considered valid, according to point-based semantics, in each interval included in valid time element. A function `VALID` is provided which extracts the valid time element from a tuple. If single intervals must be handled instead, valid time elements can be partitioned into their set of constituent intervals. This can be done by specifying option `(PERIOD)` in a `FROM`-clause. Example 4 illustrates the representation of states in *TSLQ2*.

*Example 4* The following *TSLQ2* sentence creates a valid time table, `EFile`, where tuples store the name of an employee (attribute `NAME`); the department where she/he has worked or is currently working (attribute `DEPT`); and a valid time element which stores the periods where the employee in question has worked in the corresponding department.

```
CREATE TABLE EFile (NAME CHARACTER, DEPT CHARACTER)
AS VALID STATE TO MONTH
```

A possible instance for `EFile` is shown next (where predefined timestamp `FOREVER` is used to represent a future-open interval),

| NAME | DEPT | VALID TIME |
|---|---|---|
| John Roberts | Books | {'[1/2002-6/2002]', '[1/2003-FOREVER]'} |
| John Roberts | Bazar | {'[7/2002-12/2002]'} |
| Ann Smith | Bazar | {'[3/2002-5/2002]'} |

By way of example, the following *TSLQ2* query returns the list of departments where John Roberts worked, before he entered to work in the Bazar department,

```
SELECT DISTINCT T1.DEPT
FROM EFile(PERIOD) AS T1 T2
WHERE T1.NAME = 'John Roberts' AND
      T2.NAME = T1.NAME AND
      T2.DEPT = 'Bazar'
AND VALID(T1) PRECEDES VALID(T2)
```

Notice in the previous query (example 4 above), that the construct `EFile(PERIOD)` in the `FROM`-clause, decomposes every valid time element into single periods, and so the query is actually evaluated with respect to the following table representation,

| NAME | DEPT | VALID TIME |
|---|---|---|
| John Roberts | Books | '[1/2002-6/2002]' |
| John Roberts | Books | '[1/2003-FOREVER]' |
| John Roberts | Bazar | '[7/2002-12/2002]' |
| Ann Smith | Bazar | '[3/2002-5/2002]' |

## 4.5 Discussion

**Relative information in TSQL2.** So far we have seen that *TSQL2* deals with intervals, interval relationships and states where the information is *absolute*, in the sense that interval boundaries are explicit and known only with respect to the corresponding fact. Interestingly enough, *TSQL2* also provides a limited way to model situations where the temporal validity of certain facts is only known to be *relative* to some other information in the database.

In *TSLQ2*, relative timestamps can be constructed by adding or subtracting time spans to/from other timestamps. Time spans are represented by the `INTERVAL` data type, e.g. `INTERVAL '3' MONTH` denotes a time span of 3 months. *TSLQ2* also provides a way to construct new states from other states, as it is illustrated by example 5 below,

*Example 5* The next sentence adds a new tuple to the table `EFile` (see example 4 again), which denotes that Mike Thompson started to work for the Toys department two months after John Roberts started to work for Books, and that Mike Thompson worked for Toys until Ann Smith stopped working for the Bazar department.

Notice, in the specification of this state, the absence of absolute temporal references; all we know is that the working period of Mike Thompson is relative to the first working period of John Roberts at Toy, and the last period of Ann Smith at Bazar. The *TSLQ2* sentence which inserts this state in `EFile`, is as follows,

```
INSERT INTO EFile
SELECT ’Mike Thompson’,’Toys’
VALID PERIOD
      (BEGIN(VALID(J))+INTERVAL ’2’ MONTH,
       END(LAST(VALID(D)))
FROM EFile AS J A
WHERE J.NAME = ’John Roberts’ AND
      J.DEPT = ’Books’ AND
      A.NAME = ’Ann Smith’ AND
      A.DEPT = ’Bazar’
```

Relative references may also appear in queries. For example, the following *TSLQ2* query returns the department for which John Roberts was working, six months after Mike Thompson started to work for Toys,

```
SELECT J.DEPT
FROM EFile AS J M
WHERE J.NAME = ’John Roberts’ AND
      M.NAME = ’Mike Thompson’ AND
      M.DEPT = ’Toys’ AND
      VALID(J) CONTAINS
      (BEGIN(FIRST(VALID(M))) + INTERVAL ’6’ MONTH)
```

where the predefined function `CONTAINS` checks for temporal inclusion between intervals.

We have seen that contexts where the temporal information is relative, can be modelled naturally in *Templog* (see, e.g. Figure 1). However, compared with *Templog*, the facilities provided in *TSLQ2* to handle this kind of situations is limited. Generally speaking, and as the following example shows, the *TSLQ2* constructs which allows for the representation of relative information, are best regarded as syntactic facilities. In particular, if the base information changes, then the relative facts are not updated to reflect the changes. Instead, the user is supposed to check for possible inconsistencies and updated the necessary information.

*Example 6* Consider the simple *Templog* program,

```
start_work(m1)
□( ○ start_work(m2) ← start_work(m1))
□( ○² stop_work(M) ← start_work(M))
```

which denotes that machine `m1` started working initially, that `m2` always starts working 1 time-unit after `m1` starts, and that both machines always works for 2 time-units. Clearly, information is given relative to the time when machines start working, and in particular to the time when `m1` starts.

Now assume the representation of the same information in *TSLQ2*, in which we assume a valid time table `Machines` denoting the working periods of machines.

Assume, as well, that the table has only one data attribute, `NAME`, that the granularity is given in days, and that the tuple corresponding to the working period of machine `m1` has already been inserted in the table. Then, the following sentence will add a tuple corresponding to machine `m2`, whose working period (as we have seen) is relative to the start of `m1` ( `BEGIN(VALID(M1))`).

```
INSERT INTO Machines
SELECT 'm2'
VALID PERIOD ( BEGIN(VALID(M1))+INTERVAL '1' DAY,
               BEGIN(VALID(M1))+INTERVAL '2' DAY )
FROM Machines AS M1
WHERE M1.NAME = 'm1'
```

Now, consider a change in the original starting time for `m1`, in which this happens 3 days later than initially asserted. This update is easy in *Templog*, yielding a new program,

$\bigcirc^3$`start_work(m1)`
$\Box($ $\bigcirc$ `start_work(m2)` $\leftarrow$ `start_work(m1))`
$\Box($ $\bigcirc^2$ `stop_work(M)` $\leftarrow$ `start_work(M))`

It is not difficult to note that the working period of the second machine, `m2`, is kept consistent with this new information. However, this will not be the case in our *TSLQ2* table, `Machines`. The problem is, that a clause like,

```
VALID PERIOD ( BEGIN(VALID(M1))+INTERVAL '1' DAY,
               BEGIN(VALID(M1))+INTERVAL '2' DAY )
```

is evaluated at insertion time, and so an expression such as,

```
BEGIN(VALID(M1))+INTERVAL '1' DAY
```

returns, actually, just an absolute temporal reference. Therefore, changing the valid time of the tuple `M1` will not affect the timestamps stored for `'m2'`.

**TSQL2 and TQuel.** *TQuel* (Snodgrass 1987; Tansel et al 1993) is a minimal extension to *Quel*, the relational query language for the system Ingres (Stonebraker et al 1976). It supports user-defined, valid and transaction times. The temporal structure is also similar to that of *TSQL2*: a linear, discrete, and bounded set of chronons. Timestamps are explicit and may be specified in different granularities. However, a few differences exist between *TSQL2* and *TQuel* which may impact on the modelling of intervals, interval relationships and states.

One of these differences is in the modelling of interval relationships; these are more difficult to express in *TQuel*, as the relational operator $<$ to compare timestamps is not directly available in the language. Instead, the predefined *TQuel* operator `precedes` implements $\leq$, and so negation (`not`) has to be used in conjunction to get the proper semantics for Allen's relationships. Table 2 shows the definition of

**Table 2** Interval relationships in *TQuel*.

| Interval relationship | Equivalent *TQuel* expression |
|---|---|
| *before(I,J)* | `I precede J and not (end of I equal begin of J)` |
| *meets(I,J)* | `end of I equal begin of J` |
| *overlaps(I,J)* | `begin of I precede begin of J and`<br>`begin of J precede end of I and`<br>`end of I precede end of J and`<br>`not (begin of I equal begin of J) and`<br>`not (end of I equal end of J)` |
| *during(I,J)* | `begin of J precede begin of I and`<br>`end of I precede end of J and`<br>`not (I equal J)` |
| *starts(I,J)* | `begin of I equal begin of J and`<br>`end of I precede end of J and`<br>`not (I equal J)` |
| *finishes(I,J)* | `begin of J precede begin of I and`<br>`end of I equal end of J and`<br>`not (I equal J)` |
| *equals(I,J)* | `I equal J` |

Allen's relationships in *TQuel*: *I* and *J* denote intervals, and functions `begin of` and `end of` return the timestamps of a given interval. Notice, in contrast with table 1, that expressions are not as straightforward as in *TSQL2*.

Probably the most important difference between *TQuel* and *TSLQ2*, is the representation of states with multiple validity intervals. Unlike *TSQL2* with valid time elements, *TQuel* does not handle interval sets as single entities. Then, tuples in valid time databases are assigned only a single validity interval, and modelling states with multiple intervals would require one tuple for every such interval. This does not only cause redundancy in data attributes, but also makes certain queries more difficult to express. In particular, this is the case when the query works with some, but not all, validity intervals related to the same state. Aggregate functions would be needed to search over all tuples (which may result in nested queries), just to collect the relevant intervals.

This issue occurs in the following *TQuel* query, which returns the department for which John Roberts was working, six months after Mike Thompson started to work for Toys (introduced for *TSQL2* in example 5),

```
range of J is Efile
range of M is Efile
range of T is Efile
retrieve (J.Dept)
where J.Name = "John Roberts" and
      M.Name = "Mike Thompson" and
      M.Dept = "Toys" and
      J overlaps (begin of M + %6 month%) and
      M equals earliest(T where
                          T.Name="Mike Thompson" and
                          T.Dept = "Toys")
```

where `earliest` is an aggregate function which retrieves the first of all of those intervals where Mike Thompson used to work at Toys (tuple `T`). If we compare this with the equivalent query in *TSQL2*, which we revisit below, we will notice that nested queries are not necessary, as the set of relevant intervals is already available when the tuple (`M`) is found.

```
SELECT J.DEPT
FROM EFile AS J M
WHERE J.NAME = ’John Roberts’ AND
      M.NAME = ’Mike Thompson’ AND
      M.DEPT = ’Toys’ AND
      VALID(J) CONTAINS
      (BEGIN(FIRST(VALID(M))) + INTERVAL ’6’ MONTH)
```

Notice that the construct,

```
FIRST(VALID(M))
```

has the same purpose that the aggregate function `earliest` in the *TQuel* query, i.e. to return the first intervals related to Mike Thompson working for Toys. However, in the *TSQL2* query these intervals are already available as the valid time element of `M` (and so are returned by `VALID(M)`). On the other hand, in *TQuel* a nested query is needed to collect all these intervals,

```
T where T.Name="Mike Thompson" and T.Dept="Toys"
```

## 5 Conclusions

We think the main contribution of this paper is in evaluating well known temporal query languages, such as *Templog* and *TSQL2*, from a novel perspective. Surveys on formal expressiveness of temporal query languages have populated the literature; however, the issue of how naturally valid time information can be modelled, has been mostly overlooked. To complement existing accounts, *Templog* and *TSQL2* have been evaluated from a different perspective, based upon the concepts of time intervals, interval relationships and states.

We have worked on the hypothesis that these concepts are general enough to represent a wide class of valid time information. Therefore, evaluating the languages (and related data models) with respect to these concepts gives an idea of the kind of issues which may arise in practice, in commonly found modelling tasks. In addition, and particularly in those cases in which the language does not naturally model certain concept, we have offered possible ways in which this can be achieved.

Deductive languages are not generally expected to deal with absolute temporal references. We have, therefore, assessed the modelling of interval, interval relationships and states in more natural contexts, e.g. in those where the application will typically need to reason about relative facts, or periodic information. Also, the fact that a deductive language such as *Templog* does not support intervals as primitive temporal elements, is a hindrance for the modelling of states. In addition, and among other issues, we have elaborated on the differences between *Templog* and *Datalog*$_{1S}$ when modelling states. This is interesting as reveals modelling limitations which are not obvious under different evaluation contexts.

On the other hand, relational query languages such as *TSQL2* will efficiently deal with absolute intervals (and consequently, relationships and states), as intervals are primitive blocks of valid time information. Nevertheless, we have shown that contexts in which relative information play a crucial role, might not be so straightforward to deal with. We have also compared *TSQL2* with *TQuel*, in order to reveal how small differences in the languages data models and predefined operators, may have important consequences in practice when modelling states.

We conclude this paper by pointing out further research. We believe that a promising line of research may consider extending the expressiveness criterion in order to cover other kinds of temporal information, such as events and processes (Galton 2005). One work conducted in this area is that of Terenziani (Terenziani 2000). He addressed some problems in *TSQL2* regarding the modelling of *telic facts*, i.e., facts can be valid over intervals but they are not considered valid at any subinterval in question.

This work also complements previous reports in the technical literature (Galton and Augusto 2002; Gómez and Augusto 2004) rasing awareness in the Databases and AI communities about the potential benefits that considering the mutual approaches may bring to each other. In an era of specialisation, there is a potential danger of the areas becoming ”too introspective”. We have observed that phenomenon regarding temporal concepts in AI and Databases. This article, as well as other previously mentioned, are part of an effort to encourage interaction between the areas and to increase the benefits deriving from each others’ findings.

## References

Abadi M, Manna Z (1989) Temporal Logic Programming. Symbolic Computation 8:277 – 295

Allen JF (1983) Maintaining Knowledge about Temporal Intervals. Communications of the ACM 26, No. 11:832 – 843

Allen JF (1984) Towards a General Theory of Action and Time. Artificial Intelligence 23:123 – 154

Artale A, Fisher M, Theodoludis B (eds) (2002) Proceedings of the Ninth International Workshop on Temporal Representation and Reasoning, IEEE Computer Society Press, Manchester, UK

Baudinet M (1989) Temporal logic programming is complete and expressive. In: Sixteenth ACM Symposium on Principles of Programming Languages, Austin, Texas, pp 267–280

Baudinet M (1992) A simple proof of completeness of temporal logic programming. In: nas del Cerro LF, Penttonen M (eds) Intensional Logics for Programming, Oxford University Press, pp 50–83

Baudinet M (1995) On the Expressiveness of Temporal Logic Programming. Information and Computation 117(2):157–180

Baudinet M, Niezette M, Wolper P (1991) On the representation of infinite temporal data and queries (extended abstract). In: PODS '91: Proceedings of the tenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, ACM Press, pp 280–290

Baudinet M, Chomiki J, Wolper P (1993) Temporal Deductive Databases. In: Tansel A, Clifford J, Gadia S, Jajodia S, Segev A, Snodgrass R (eds) Temporal Data Bases (Theory, Design and Implementation), The Benjamin Cummings Pub. Co., California

Bettini C, Montanari A (eds) (2001) Proceedings of the Eigth International Workshop on Temporal Representation and Reasoning, IEEE Computer Society Press, Cividale del Friuli, Italy

Bettini C, Dyreson CE, Evans WS, Snodgrass RT, Wang XS (1998a) A Glossary of Time Granularity Concepts. In: Etzioni O, Jajodia S, Sripada S (eds) Temporal Databases: Research and Practice, Springer Verlag

Bettini C, Wang XS, Jajodia S (1998b) Temporal semantic assumptions and their use in databases. IEEE Transactions on Knowledge and Data Engineering 10(2)

Bohlen M, Chomicki J, Snodgrass RT, Toman D (1996) Querying TSQL2 Databases with Temporal Logic. In: Proceedings EDBT 96, Lecture Notes in Computer Science 1057, pp 325–341

Cervesato I, Franceschet M, Montanari A (2000) A guided tour through some extensions of the event calculus. Computational Intelligence 16(2):307–347

Chittaro L, Montanari A (1996) Efficient temporal reasoning in the cached event calculus. Computational Intelligence 12(3):359–382

Chomicki J (1990a) Functional Deductive Databases: Query Processing in the Presence of Limited Functional Symbols. PhD thesis, Rutgers University, New Brunswick, New Jersey

Chomicki J (1990b) Polynomial time query processing in temporal deductive databases. In: Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pp 379–391

Chomicki J (1994) Temporal query languages: a survey. In: Proceedings of ICTL'94, LNCS, vol 827, Springer-Verlag, pp 506–534

Clifford J, Dyreson CE, Isakowitz T, Jensen CS, Snodgrass RT (1997) On the semantics of "now" in databases. ACM Transactions on Database Systems 22(2):171–214

Cobo ML, Augusto JC (1999) EMTPL: A Programming Language for Temporal Deductive Data Bases. In: Proceedings de la XIX International Conference of the Chilean Computer Science Society, Talca, Chile, pp 170–178

Dowty D (1986) The effects of the aspectual class on the temporal structure of discourse. Linguistics and Philosophy 9(1):37–61

Dyreson CE, Snodgrass RT (1998) Supporting valid-time indeterminacy. ACM Transactions on Database Systems 23(1):1–57

Etzioni O, Jajodia S, Sripada S (eds) (1998) Temporal Databases: Research and Practice. Springer-Verlag

Freksa C (1992) Temporal reasoning based on semi-intervals. Artificial Intelligence 54(1):199–227

Gallaire H, Minker J, Nicolas JM (1984) Logic and databases: A deductive approach. ACM Computing Surveys 16(2):153–185

Galton A (2005) Eventualities. In: M Fisher DG, Vila L (eds) Handbook of Temporal Reasoning in Artificial Intelligence, Elsevier

Galton A, Augusto JC (2002) Two approaches to event definition. In: A Hameurlain RC, Traunmüller R (eds) Proceedings of 13th International Conference on Database and Expert Systems Applications (DEXA 2002), Springer-Verlag, Aix-en-Provence, France, pp 547–556

Gómez RS, Augusto JC (2000) Un Análisis comparativo de Lenguajes de Consulta para Bases de Datos Temporales. In: Proceedings del VI Congreso Argentino de Cs. de la Computación, CACiC2000, Usuahia, 2 al 7 de octubre de 2000, pp 111–122

Gómez RS, Augusto JC (2004) Durative event composition in active databases. In: Proceedings of 6th International Conference on Enterprise Information Systems, INSTICC Press, Porto, Portugal, vol 1, pp 306–311

Goodwin S, Trudel A (eds) (2000) Proceedings of the Seventh International Workshop on Temporal Representation and Reasoning, IEEE Computer Society Press, Cape Breton, Canada

Grant J, Minker J (1992) The impact of logic programming on databases. Communications of the ACM 35(3):66–81

Hamblin CL (1972) Instants and Intervals. In: J Fraser FH, Muller G (eds) The Study of Time, Springer Verlag, New York, pp 324–328

Jensen CS, Snodgrass RT (1996) Semantics of Time-varying information. Information Systems 21(4):311–352

Jensen CS, Dyreson CE, Bohlen M, Clifford J, Elmasri R, Gadia SK, Grandi F, Hayes P, Jajodia S, Kafer W, Kline N, Lorentzos N, Mitsopoulos Y, Montanari A, Nonen D, Peressi E, Pernici B, Roddick JF, Sarda NL, Scalas MR, Segev A, Snodgrass RT, Soo MD, Tansel A, Tiberio P, Wiederhold G (1998) A Glossary of Time Granularity Concepts. In: Etzioni O, Jajodia S, Sripada S (eds) Temporal Databases: Research and Practice, Springer Verlag

Kowalski R (1992) Database updates in the event calculus. Journal of Logic Programming 12:121–146

Kowalski R, Sergot M (1986) A logic-based calculus of events. New Generation Computing 4:67–95

Lloyd JW (1987) Foundations of Logic Programming, second edition edn. Springer-Verlag, Berlin

McKenzie E, Snodgrass R (1989) An Evalution of Algebras Incorporating Time. Tech. rep., The University of Arizona

Morris R, Khatib L (eds) (1999) Proceedings of the Sixth International Workshop on Temporal Representation and Reasoning, IEEE Computer Society Press, Los Alamitos, California, USA

Revesz P (1993) A closed-form evaluation for Datalog queries with integer (gap)-order constraints. Theoretical Computer Science 116(1):117–149

Reynolds M, Sattar A (eds) (2003) Proceedings of the Tenth International Workshop on Temporal Representation and Reasoning and Fourth Interantional Conference on Temporal Reasoning, IEEE Computer Society Press, Cairns, Queensland, Australia

Snodgrass RT (1987) The Temporal Query Languaje TQuel. ACM Transactions on Database Systems 12(2):247–298

Snodgrass RT (ed) (1995) The TSQL2 Temporal Query Language. Kluwer Academic Publishers, Berlin

Snodgrass RT, Ahn I (1986) Temporal Databases. IEEE Computer 19, No. 9:35 – 42

Stonebraker M, Wong E, Kreps P, Held G (1976) The Design and Implementation of INGRES. ACM Transactions on Database Systems 1, No. 3:189 – 222

Tansel A, Tin E (1998) Expressive Power of Temporal Relational Query Languages and Temporal Completeness. In: Etzioni O, Jajodia S, Sripada S (eds) Temporal Databases: Research and Practice, Springer Verlag

Tansel A, Clifford J, Gadia S, Jajodia S, Segev A, Snodgrass RT (1993) Temporal Data Bases (Theory, Design and Implementation). The Benjamin Cummings Pub. Co., California

Terenziani P (2000) Is Point-Based Semantics Always Adequate for Temporal DataBases? In: Proceedings of the Seventh International Workshop on Temporal Representation and Reasoning (Time-00), pp 191–199

Toman D (1996) Point-based vs. Interval-based Temporal Query Languages. In: Proceedings ACM PODS 1996, pp 58–67

Toman D, Niwinski D (1996) First-Order Temporal Queries Inexpressible in Temporal Logic. In: Proceedings EDBT'96, Lecture Notes in Computer Science 1057, pp 307–324

Toman D, Chomicki J, Rogers D (1994) Datalog with integer periodicity constraints. In: ILPS '94: Proceedings of the 1994 International Symposium on Logic programming, MIT Press, pp 189–203