A non-standard temporal deductive database system

Jean-Raymond Gagné  John Plaice
Département d’informatique, Université Laval,
Ste-Foy (Québec) Canada G1K 7P4
\{gagne,plaice\}@ift.ulaval.ca.
1996

Abstract
A new temporal deductive database system supporting a non-standard model of time is introduced. It consists of Non-standard Temporal Datalog (nstl) and Non-standard Temporal Relational Algebra (nтра). The time line consists of non-standard reals that are of the form \( \langle r, z \rangle \), where \( r \in \mathbb{R} \) and \( z \in \mathbb{Z} \), using the natural order. Each real \( r \) determines a macro-instant, and each pair \( \langle r, z \rangle \) defines a micro-instant. The set of macro-instants forms a dense order, thereby allowing different relations to be valid at different moments, with independent rates of evolution. The micro-instants ensure that all intervals are closed, thereby simplifying the semantics. At the same time, it becomes possible to define a discrete memory operator.

The \( \text{nstl} \) language is an extension of Datalog in which the fact base is augmented with interval timestamps and in which rules are an extension of generalized Horn clauses that allow a memory operator “;” and allow timestamped atoms in the body.

The \( \text{nтра} \) language is a pointwise extension of the relational algebra over the time line. To do this, three temporal operators are added to the relational algebra.

1 Introduction

One of the key problems in temporal database design has to do with the choice of domains for time. This problem, commonly referred to as the granularity of time, defines the kind of temporal information that can be stored in a database and the kinds of operations that can be applied to this information.

Wiederhold et al. (1991) show that a temporal database can easily contain data based on different granularities (e.g., days, hours, picoseconds, etc.). As a result, they conclude that these different granularities can only be resolved by using a finest granularity, probably the real number line. Since the real numbers form a dense set, one can always define a finer granularity if such is needed.

However, databases are not just used for storing information, rather computations are effected on data therein. The order in which these computations takes place can be significant, which means that the time line must be able to take them into account.

A question therefore arises: What is the granularity for computation? If we take some positive real-numbered value, then our model is no longer capable of dealing with arbitrary granularity. So it would appear that the other possibility is zero. However, it then becomes difficult to distinguish different states within the same computation.

The solution to this problem comes from non-standard analysis, invented by Newton and Leibniz in the seventeenth century and formalized by Abraham Robinson (1974). What we need are positive infinitesimal values that are strictly smaller than any positive real number, while still strictly greater than 0.
This paper presents a non-standard temporal database system, in which the time line consists of non-standard reals (Nelson, 1977; McLaughlin and Miller, 1992; Robinson, 1974), which are, in this case, pairs \((r, z)\), where \(r \in \mathbb{R}\) and \(z \in \mathbb{Z}\).

This approach is compatible with that taken in synchronous languages, such as Esterel (Boussinot and De Simone, 1991), Lustre (Halbwachs et al., 1991) and Signal (Le Guernic et al., 1991), used for the programming of reactive systems. These languages assume that reactions to input events are instantaneous. Nevertheless, the ordering of computations within a single instant is of utmost importance.

Similarly, work in hybrid systems, which combine continuously changing variables with variables that change at discrete instants, makes similar suppositions. In fact, the non-standard approach that we first took in the preliminary version of this paper (Gagné and Plaice, 1995) has been independently developed by Iwasaki et al. (1995) for the analysis of hybrid systems.

Choosing our non-standard time line has many advantages. First, we get a dense set of macro-instants, which allows for the use of any granularity of time. Second, within each macro-instant, we have available a discrete set of micro-instants. In a certain sense, we have “discretized” the reals explicitly: the result is that all intervals are both open and closed, so we can refer to the “previous instant” and the “following instant”, without any ambiguity.

Any other approach, using some positive real value as an approximation of infinitesimal values would truly create a discrete set, losing the dense property of the reals.

The remaining question is how to combine or synchronize data based on unsynchronized granularities. It turns out that this problem has already been studied in the domain of synchronous languages. In particular, the authors are developing a language called BlizzarD, using non-standard analysis, to furnish the semantics of reactive systems.

Blizzard is a dataflow language, where operators apply to operands in a pointwise manner, along with an additional memory operator, called before. These operators are used below to extend the temporal algebra into the nstra.

To develop a deductive database system, we use Orgun and Wadge’s technique of combining an extension of Datalog (Orgun and Wadge, 1992), nstl, with an extension of the temporal algebra, nstra. The basic extension in both formalisms is the addition of a memory operator, before in nstra and “;” in nstl.

It should be understood that none of the BlizzarD operators can refer to the future, nor can they change the past. So, this approach is well suited to simulating real time, even though we are working with valid time databases.

This paper presents nstl and nstra, along with their interaction. Section 2 describes the underlying time domain and section 3 specifies the syntax of nstl formulas, Horn rules and programs. Their semantics are given in section 4, using an appropriate adaptation of the perfect-model approach (Przymusinski, 1988). Section 5 shows that nstl programs are computable, by defining a revised version of the \(T_P\) operator, used to compute a non-standard temporal Herbrand model of an nstl program. This model is proven in section 6 to be the least one. Section 7 presents the BlizzarD language, which is used in section 8 to extend the relational algebra to nstra. The interaction between nstra and nstl is also given in this section. Finally, we present our conclusions.

2 Time domain

We begin by formalizing the concepts of macro-instant and micro-instant, along with the relationship between the two. These concepts, along with the others that are formalized below, are all summarized by Figure 1.

**Definition 1.** The time domain is the set \(T = \mathbb{R} \times \mathbb{Z}\), endowed with the natural lexicographic order. Each real \(r\) determines a macro-instant and each pair \(t = (r, z)\), where \(r \in \mathbb{R}\) and \(z \in \mathbb{Z}\), defines a micro-instant. The two components of \((r, z)\) are accessed through projection functions \(\pi_R\) and \(\pi_Z\), such that \(r = \pi_R(r, z)\) (the real part) and \(z = \pi_Z(r, z)\) (the integer part). A macro-instant \(r\) consists of all micro-instants whose real part is \(r\).
A time domain is only useful if an order can be imposed on that domain. The order we use is the natural lexicographical order.

**Definition 2.** Let \( t_0 = (r_0, z_0) \) and \( t_1 = (r_1, z_1) \) be two micro-instants in \( T \). Then \( t_0 \leq t_1 \) if and only if \( r_0 < r_1 \) or \( r_0 = r_1 \) and \( z_0 \leq z_1 \).

Once we have an order, the definition for intervals follows automatically.

**Definition 3.** Let \( t_0 \) and \( t_1 \) be micro-instants in \( T \). Then \( [t_0, t_1] \) defines an interval over \( T \). For all \( t \in T \) such that \( t_0 \leq t \leq t_1 \), we have \( t \in [t_0, t_1] \). Any micro-instant \( t \) may also be written \([t, t]\). An arbitrary interval is normally written as \( \Delta \).

We are not interested in modeling all of eternity. Rather, in any given base, we are interested in a particular subset of the time line. Rather than using some special value 0, which gives the impression that at some point time was invented, we allow the use of arbitrarily defined starting and ending points.

**Definition 4.** The names start and now refer, respectively, to the first and last micro-instants that are relevant to an application. The interval \( U_T = [\text{start}, \text{now}] \), called the universal set of instants, contains all instants of interest.

To model the valid time of a formula, we need to manipulate sets of intervals, here called temporal elements. Note that empty temporal elements (empty unions) will correspond to formulas that are never true.

**Definition 5.** A temporal element \( I \) is a finite union of intervals in \( U_T \). This set of intervals is closed under finite applications of union, intersection, and complementation. With \( U_T \) as its maximum element, and the empty set as its minimum element, it forms a Boolean algebra. An interval \( \Delta \in I \) is maximal if there does not exist another \( \Delta' \in I \) such that \( \Delta \cup \Delta' \) is an interval.

Once temporal elements are defined, it becomes clear that only certain micro-instants — the relevant micro-instants — are of importance in deriving new information.

**Definition 6.** The relevant micro-instants of an interval \([t_0, t_1]\) are simply \( t_0 \) and \( t_1 \). The set of relevant micro-instants of a temporal element \( I \) is the set

\[
\Lambda(I) = \{ t \mid t \text{ is a relevant micro-instant of a maximal interval of } I \}.
\]

To ensure computability, the set of relevant micro-instants in a given interval must be finite. Below is a simple statement of when this is true.

**Proposition 1.** Let \( I \subseteq U_T \). Then \( I \) is a temporal element iff \( \Lambda(I) \) is finite.

A proof can be found in (Gadia, 1988).
3 Syntax

Logic programs without any function symbols, i.e. DATALOG programs (Ullman, 1988), can be regarded as deductive databases. Since DATALOG is function-free, all predicates in a DATALOG program are guaranteed to represent finite relations: the domain of any Herbrand interpretation of the program is finite.

For the purposes of the following definitions, we assume the existence of a finite signature Σ with constant symbols, predicate symbols (each of a given arity), and with no function symbols. We also assume the existence of a countably infinite set $V$ of variables. We write $T_{\Sigma}[V]$ for the set of terms (including variables) over $\Sigma$ and $At_{\Sigma}$ for the set of atoms over $\Sigma$.

**Definition 7.** Let $p \in \Sigma$ be a predicate symbol, $n$ be the arity of $p$ and $x_1, \ldots, x_n$ be terms in $T_{\Sigma}[V]$. Then $p(x_1,\ldots,x_n)$ is a simple atomic formula (or simple atom). To simplify the presentation below, $p(x_1,\ldots,x_n)$ will often be abbreviated as $p(\vec{x})$ or even just as $p$.

**Definition 8.** Let $\Delta$ be an interval in $U_T$ and $p$ a simple atom. Then $\Delta p$ is a timestamped atom, and $\Delta$ is called a timestamp. The meaning of $\Delta p$ is that $p$ is valid during the interval $\Delta$.

**Definition 9.** An nstl atom is either a simple atom or a timestamped atom. An arbitrary nstl atom is written $\alpha$.

**Definition 10.** The set of nstl formulas is given by the following grammar:

$$\varphi ::= \alpha | \neg \alpha | \varphi \lor \varphi | \varphi \land \varphi | \varphi ; \varphi | (\forall X)\varphi$$

where $\varphi$ stands for an arbitrary nstl formula.

The other logical connectives are composed as usual ($\varphi \leftarrow \psi$ is equivalent to $\varphi \lor \neg \psi$ and $(\exists X)\varphi$ is equivalent to $\neg (\forall X)\neg \varphi$). Intuitively, $\Delta p$ means "$p$ is valid during the interval $\Delta$" and $\varphi; \psi$ means "$\varphi$ is true at some time strictly before $\psi$ is true".

**Definition 11.** The set of nstl conjunctions is given by the following grammar:

$$\chi ::= \alpha | \neg \alpha | \chi \land \chi | \chi ; \chi$$

where $\chi$ is an arbitrary nstl conjunction.

In other words, an nstl conjunction is a special case of formula that does not use disjunction or universal quantifiers.

**Definition 12.** An nstl Horn rule is a formula of the form $\alpha \leftarrow \chi$, where $\alpha$ is an nstl atom and $\chi$ is an nstl conjunction.

**Definition 13.** An nstl program is a set of nstl Horn rules.

4 Perfect-model semantics

The semantics of nstl programs is derived from the standard perfect-model semantics of ordinary logic programs. To ensure that this is possible, the Herbrand base must be modified so that it can take into account when predicates are valid. Therefore, facts are not just ground atoms but, rather, pairs consisting of a temporal element and a formula.

First, let us redefine the notions of $\Sigma$-interpretation, validity, models and Herbrand models in the context of nstl. (See (Lloyd, 1987) for a full discussion of ordinary logic programs and (Przymusinski, 1988) for a full treatment of perfect-model semantics.) As in section 3, we assume the existence of a function-free signature $\Sigma$.

Interpretations of nstl programs must include a temporal element for each ground atom. Doing so defines the valid time for that formula.

**Definition 14.** An nstl $\Sigma$-interpretation $A$ consists of:
1. a non-empty set of constants called $\text{dom}(A)$;
2. an element $c_A \in \text{dom}(A)$ for each constant $c \in \Sigma$;
3. a relation $p_A \subseteq \text{dom}(A)^n$ for each $n$-ary predicate symbol $p \in \Sigma$;
4. a temporal element $I_\alpha \subseteq T \,$ for each $\alpha \in \text{At}_\Sigma$.

The definition of a Herbrand interpretation is standard.

**Definition 15.** An nstl Herbrand interpretation is a $\Sigma$-interpretation for which $\text{dom}(A)$ is the set of all ground atoms of $\Sigma$.

Now comes the interesting part. Facts for nstl are (temporal element, atom) pairs. Each atom has a valid time, defining when it is true.

**Definition 16.** Let $I$ be a temporal element and $\alpha$ be a ground simple (no timestamp) atom. Then the pair $I \alpha$ is an nstl fact.

**Definition 17.** An nstl Herbrand base is a set of nstl facts, one for each ground simple atom $\alpha \in \text{At}_\Sigma$. This base states when each of these formulas is valid.

Herbrand bases are interesting in general because they have nice simple properties. These properties are shared by nstl Herbrand bases.

**Proposition 2.** The set of relevant timestamps in an nstl Herbrand base is finite.

**Proof.** Follows immediately from the fact that the standard Herbrand base is finite. \hfill $\Box$

**Proposition 3.** There is an nstl Herbrand base corresponding to each nstl Herbrand interpretation.

**Proof.** Let $p_H$ be the relation $p$ in the nstl Herbrand interpretation $H$. We can construct the nstl Herbrand base $H_b$ from an nstl Herbrand interpretation $H$:

$$H_b = \{ I p(\bar{x}) \mid p \in \Sigma, \bar{x} \in p_H \text{ and } I \text{ is the temporal element corresponding to } p(\bar{x}) \in \text{At}_\Sigma \}$$

Inversely, the nstl Herbrand interpretation $H$ may be constructed from an nstl Herbrand base $H_b$ if we interpret every $n$-ary predicate symbol $p$ in $\Sigma$ by:

$$p_H = \{ \bar{x} \mid I p(\bar{x}) \in H_b \}$$

and for every atom $p(\bar{x}) \in \text{At}_\Sigma$, $I_{p(\bar{x})}$ is equal to the temporal element associated to $p(\bar{x})$ in $H$. \hfill $\Box$

Since we can either use the nstl Herbrand base or the nstl Herbrand interpretation without loss of information, we may abuse language and write $H$ in place of $H_b$.

To simulate real-time and to ensure computability over the entire time line, neither changing the past nor querying the future is allowed. The past is done and the future has not yet taken place.

**Definition 18.** Let $H$ be an nstl Herbrand interpretation, $t$ be a micro-instant, and $\varphi$ be an nstl formula, we define the validity of $\varphi$ in interpretation $H$ at the micro-instant $t$, denoted $H \models_t \varphi$ as:

- $H \models_t p(\bar{x})$ \iff $\exists I p(\bar{x}) \in H$ such that $t \in I$,
- $H \models_t [t_0, t_1] \varphi$ \iff $t_1 \leq t$ and $(\forall t' \in [t_0, t_1]) H \models_{t'} \varphi$,
- $H \models_t \neg \varphi$ \iff $H \not\models_t \varphi$,
- $H \models_t (\varphi \land \psi)$ \iff $H \models_t \varphi$ and $H \models_t \psi$,
- $H \models_t (\varphi \lor \psi)$ \iff $H \models_t \varphi$ or $H \models_t \psi$,
- $H \models_t (\varphi ; \psi)$ \iff $H \models_t \psi$ and $(\exists t' < t) H \models_{t'} \varphi$,
- $H \models_t (\forall X) \varphi$ \iff for every $x \in T_\Sigma, H \models_t \varphi[X/x]$,

where $\varphi[X/x]$ stands for the substitution of term $x$ for free occurrences of variable $X$ in formula $\varphi$. 

5
Definition 19. Let \( P \) be an NSTL program, \( H \) be an NSTL Herbrand interpretation, and \( t \) be a micro-instant. Then \( H \) is a model of \( P \) iff for all \( \varphi \) in \( P \) and for all \( t \in U_T \), \( H \models t \varphi \).

The NSTL includes negation. The NSTRA “default” operator (see section 7), when translated into NSTL, explicitly uses negation, which may be interpreted as negation as failure. The semantics is extended using perfect-model semantics (Przymusinski, 1988).

As we extended the usual Horn clause admitting negation with serial conjunction (“;”) and timestamped atoms, we now introduce the notion of the actual part of a body of an NSTL Horn rule.

Definition 20. The actual part of an NSTL conjunction \( \chi \) is \( \Upsilon(\chi) \), defined case by case as follows:

\[
\begin{align*}
\Upsilon(p) &= \{p\}, \quad p \text{ is an atom}, \\
\Upsilon(\Delta p) &= \{p\}, \\
\Upsilon(\neg p) &= \{-p\}, \\
\Upsilon(\neg \Delta p) &= \{-p\}, \\
\Upsilon(\chi_1 \land \chi_2) &= \Upsilon(\chi_1) \cup \Upsilon(\chi_2), \\
\Upsilon(\chi_1 ; \chi_2) &= \Upsilon(\chi_2).
\end{align*}
\]

Definition 21. The actual part of a body of an NSTL Horn rule \( \alpha \leftarrow \chi \) is the set \( \Upsilon(\chi) \).

The semantics of negation as failure can be problematic when recursion occurs through the use of negation. To avoid this problem, only locally stratified programs are considered (Przymusinski, 1988).

The stratification only takes place over a micro-instant, so only the actual part of bodies of NSTL Horn rules are examined. We begin by defining local stratification.

Definition 22. Let \( P \) be an NSTL program. Then \( P^* \) denotes the ground instantiation of \( P \) (the set of all ground instances of rules in \( P \)). Then we construct a directed graph \( D(P) \), whose nodes are atoms in \( P^* \). The arcs in the graph are defined as follows:

- There is an ordinary arc from \( q \) to \( p \) iff \( q \) occurs in the actual part of a body of a rule in \( P^* \) whose head contains \( p \).
- There is a negative arc from \( q \) to \( p \) iff \( \neg q \) occurs in the actual part of a body of a rule in \( P^* \) whose head contains \( p \).

Definition 23. Let \( p \) and \( q \) be ground NSTL simple atoms. Then \( p \preceq q \) iff there is a directed path from \( q \) to \( p \).

Definition 24. Let \( p \) and \( q \) be ground NSTL simple atoms. Then \( p \preceq q \) iff there is a directed path from \( q \) to \( p \) with at least one negative arc.

Definition 25. An NSTL program \( P \) is locally stratified if and only if the relation “\( \preceq \)” is well-founded, i.e. it has no infinite sequence of atoms \( p_1, p_2, \ldots \) such that \( p_1 \preceq p_2 \preceq \ldots \).

The perfect models of an NSTL program \( P \) are defined in terms of a preference order \( \ll \) over the models of \( P \).

Definition 26. The expression \( M \ll M' \) holds iff for any atom \( \alpha \) and any micro-instant \( t \), if \( M \models t \alpha \) and \( M' \not\models t \alpha \), then there exists \( \beta \) such that \( \beta \preceq \alpha \), \( M \models t \beta \) and \( M' \not\models t \beta \).

Definition 27. Let \( P \) be an NSTL program. Then a perfect model of \( P \) is any model that is minimal with respect to \( \ll \). We write \( \mathcal{M}(P) \) for the set of perfect models of \( P \).

Below, we assume that entailment includes negation as failure and we only consider perfect models.
5 The computation process of an NSTL program

The NSTL Herbrand base is computed iteratively. The base is computed at each relevant micro-instant, starting from the base computed in the previous relevant micro-instant. The process begins at the first relevant micro-instant, with an empty base. The process finishes with the last relevant micro-instant, and the result is the NSTL Herbrand base. Within each micro-instant, entailment is computed through negation by failure.

5.1 Fixpoint computation for a specific micro-instant

In the presentation of fixpoint computation, we make use of the following well-known theorem (Tarski, 1955).

**Proposition 4** (Tarski-Knaster). Let \( G \) be a monotone and continuous function. Then

\[
\mu G = (G(\emptyset))^\omega = \bigcup_{n \geq 0} G^n(\emptyset).
\]

The \( \mu \) operator, when applied to a function \( G \), finds the least fixpoint of this function, by iteratively calculating

\[
X = G^0(\emptyset), G^1(\emptyset), \ldots, G^n(\emptyset),
\]

until \( G^{n+1}(X) = G^n(X) \), meaning that \( G^n(X) \) is the least fixpoint of \( G \).

The standard \( TP \) operator used for the semantics of logic programs is modified to take into account the specific micro-instant.

**Definition 28.** Let \( P \) be an NSTL program, \( t \) be a micro-instant and \( H \) be an NSTL Herbrand base.

\[
TP,t,H(H) = H \cup \{ \theta \alpha \mid \alpha \leftarrow \chi \in P \text{ and } \theta : V \rightarrow T_{\Sigma} \text{ and } (H \cup H) \models t \chi \}.
\]

**Definition 29.** The fixpoint for a micro-instant \( t \) is \( \mu TP,t,H \), where \( H \) is the Herbrand base computed at the previous micro-instant.

5.2 Computing over the set of all relevant micro-instants

The key to computing for all instants is determining all the potentially relevant micro-instants, i.e., those instants in which facts might be added to or modified in the Herbrand base. These are computed using the \( \Lambda_P(H) \) operator, which is an extension of the \( \Lambda \) operator.

**Definition 30.** The set of potentially relevant micro-instants for an NSTL rule \( \varphi \) is \( \Lambda_H(\varphi) \).

\[
\Lambda_H(p) = \{ \Lambda(I) \mid \exists \theta : V \rightarrow T_{\Sigma} \text{ and } Ip\theta \in H \},
\]

\[
\Lambda_H([t_0, t_1]p) = \{ t_1 \},
\]

\[
\Lambda_H(\neg \varphi) = \Lambda_H(\varphi),
\]

\[
\Lambda_H(\varphi \land \psi) = \Lambda_H(\varphi) \cup \Lambda_H(\psi),
\]

\[
\Lambda_H(\varphi \lor \psi) = \Lambda_H(\psi) \cup \{ t + 0, 1 \mid t \in \Lambda_H(\varphi) \},
\]

\[
\Lambda_H(p \leftarrow \chi) = \Lambda_H(\chi),
\]

\[
\Lambda_H([t_0, t_1]p \leftarrow \chi) = \{ t_0, t_1 \} \cup [t_0, t_1] \cap \Lambda_H(\chi).
\]

where the \( I \) correspond to temporal elements. Rule \( \Lambda_H([t_0, t_1]p \leftarrow \chi) \) corresponds to asserting facts.

**Definition 31.** The set of potentially relevant micro-instants for an NSTL program \( P \) for a Herbrand base \( H \) is given by

\[
\Lambda_P(H) = \bigcup_{\varphi \in P} \Lambda_H(\varphi).
\]
Definition 32. An NSTL program successively applies, in order, the fixpoint calculations for all relevant micro-instants:

\[ F_P = \bigcup_{i=0..|M|} F_{P,i}(\emptyset), \]

where

\[ F_{P,0}(H) = H, \]
\[ F_{P,i}(H) = \mu T_{P,M_i}(F_{P,i-1}(H)), \]
\[ M = \{\text{start}\} \cup [\text{start, now}] \cap \Lambda_P(\{U_T\} \times At_S). \]

This version of the function \( F \) computes a fixpoint for a micro-instant and then advances to the next relevant micro-instant. This process begins with the first relevant micro-instant, start, and terminates after the last relevant micro-instant, now.

6 Results about NSTL

Remember that \( \mathcal{M}(P) \) stands for the set of perfect models of a program \( P \). By definition we have: for all \( A \in \mathcal{M}(P) \), \( A \models P \), where \( \models \) is the stratified entailment defined in section 4.

**Theorem 1.** Let \( P \) be a NSTL program for which the directed graph \( D(P) \) is acyclic (Horn rules are not recursive). Among the NSTL Herbrand interpretations that are models of \( P \), there is a least NSTL Herbrand perfect model \( L_P \) defined by:

\[ L_P = \{Ip(x_1,\ldots,x_n) \mid x_1,\ldots,x_n \in T_\Sigma \text{ and } I \subseteq U_T \text{ and } (\forall t \in I)P \models_t p(x_1,\ldots,x_n)\}. \]

**Proof.** Let \( \alpha \leftarrow \beta_1 \land \ldots \land \beta_i \land \ldots \land \beta_m \) be an NSTL formula in \( P \). Should \( m = 0 \), then this is simply a fact. We rewrite this formula in a more practical form as

\[ \alpha \leftarrow \beta_1^{k_1} \cdots \beta_m^{k_m}, \]

where each \( k_i \) denotes the number of ';' to the right of the \( \beta_i \).

So, let \( X_1,\ldots,X_n \) be the variables in \( \alpha \) and the \( \beta_i \). Then \( L_P \) is a perfect model of \( P \) can be written as follows:

\[ L_P \models \alpha \leftarrow \beta_1^{k_1} \cdots \beta_m^{k_m}. \]

This holds if for all \( t \in U_T \),

\[ L_P \models_t \forall X_1 \ldots \forall X_n (\beta_1^{k_1} \cdots \beta_m^{k_m} \to \alpha). \]

This last statement holds if for all sequences of micro-instants

\[ t_{k_1} < t_{k_2} < \ldots < t_{k_i} < \ldots < t_{k_m} = t, \]

where each \( t_{k_i} \in U_T \), and for all ground terms \( x_1,\ldots,x_n \), then

\[ L_P \models_{t_{k_i}} \beta_i^{k_i} [X_1/x_1,\ldots,X_n/x_n], i = 1..m \Rightarrow L_P \models_{t} \alpha[X_1/x_1,\ldots,X_n/x_n] \in L_P. \]

This last implication can be translated into

\[ (\exists I_{\beta_i})_i \beta_i [X_1/x_1,\ldots,X_n/x_n] \in L_P \text{ and } t_{k_i} \in I_{\beta_i}, i = 1..m \]

\[ \Rightarrow (\exists I_{\alpha})_i \alpha[X_1/x_1,\ldots,X_n/x_n] \in L_P \text{ and } t \in I_{\alpha}; \]

which holds if

\[ P \models_{t_{k_i}} \beta_i^{k_i} [X_1/x_1,\ldots,X_n/x_n], i = 1..m \Rightarrow P \models_{t} \alpha[X_1/x_1,\ldots,X_n/x_n]; \]
which is true, since $P \models \alpha ((\beta_1 \land \ldots \land \beta_i \land \ldots \land \beta_m) \rightarrow \alpha)[X_1/x_1, \ldots, X_n/x_n]$.

Hence $L_P$ is a perfect model of $P$. We must still show that it is the least model.

Let $A \in \mathcal{M}(P)$ be an arbitrary perfect model of $P$ and $H_{L_P}$ be the nstl Herbrand interpretation corresponding to the nstl Herbrand base $L_P$. Since $A$ is a model of all formulas that follow from $P$, it must also be a model of all formulas in $H_{L_P}$. Since formulas in $H_{L_P}$ are all ground simple nstl atoms, it follows that $L_P \subseteq A$.

The set $\mathcal{M}(P)$ is closed under model-intersection, so $\cap \mathcal{M}(P) = L_P$. \hfill \qed

**Theorem 2.** The set $\cap \mathcal{M}(P)$ is computable and equal to $F_P$.

**Proof.** We first prove that $\cap \mathcal{M}(P)$ is closed under intersection. This is true for ordinary logic programs (Lloyd, 1987). What is different here is the presence of temporal elements associated to each atom. But Definition 2.5 tells us that the intersection of temporal elements is itself a temporal element. It follows that the same holds true for perfect models of $P$.

We next prove that $F_P$ is a fixpoint. We do this by proving that for all $i \geq 0$, $F_{p,i}$ is a fixpoint.

Case 1: $F_{p,0}(H) = H$ is a fixpoint.

Case 2: Suppose that $F_{p,i-1}(H)$ is also a fixpoint:

$$T_{p,M_i,(F_{p,i}(H))(\emptyset)} = T_{p,M_i,(F_{p,i-1}(H))(\emptyset)} \uparrow^\omega = T_{p,M_i,(F_{p,i-1}(H))} \uparrow^\omega = F_{p,i}(H).$$

So $F_P$ is a fixpoint. Now we must prove that it is the least fixpoint.

Suppose for contradiction that $F_P$ is not the least fixpoint. Then there must be $\alpha \in F_P$ such that $P \not\models \alpha$. However, $\alpha \in F_P$ implies that there exists $X$ such that $\alpha \in F_{p,X}(\emptyset)$ (by definition). Since $F_{p,X}(\emptyset) \subseteq F_{p}$, we have $\alpha \in F_{p}$, a contradiction. Therefore $F_P$ is the least fixpoint, hence $F_P = L_P = \cap \mathcal{M}(P)$.

Now we must show that $F_P$ is finite and computable. This is done by showing that for all $i \geq 0$, $F_{p,i}(H)$ is finite and computable.

We first show by induction that if $H$ is finite, then for all $i \geq 0$, $F_{p,i}(H)$ is finite.

Case 1: $F_{p,0}(H) = H$.

Case 2: Suppose that $F_{p,i-1}(H)$ is finite. We wish to prove that $F_{p,i}(H)$ is finite:

$$F_{p,i}(H) = T_{p,M_i,(F_{p,i-1}(H))}(\emptyset) \uparrow^\omega,$$

which is finite.

Since according to the definition of $F_P$, the initial Herbrand base is $\emptyset$, and all subsequent bases are the result of applying an $F_{p,i}$, it follows that all of the $F_{p,i}(H)$ in the definition of $F_P$ are finite.

Since the cardinality of $M$ is finite and $\mu X T_{p,i}(H)$ is computable, it follows that $F_P$ is computable. \hfill \qed

### 7 Blizzard

The basis for the work below are the primitives of BLIZZARD, a language invented to express the semantics of timestamped dataflow programming, for the purposes of reactive systems. In BLIZZARD, discrete events take place over a dense time line. Here, variables can be defined everywhere on the non-standard time line, but should only change a finite number of times in a given closed interval.

Let $V$ be any set of values. Let $U_T = [\text{start, now}]$ be the time domain as defined in section 2. A subset $T$ of $U_T$ is called a date set. A flow $X$ on $V$ dated by $U_T$ is a pair $(T_X, v_X)$, where $T_X$ is a date set and $v_X : T_X \rightarrow V$.

It is important to note that a flow can be finite, even empty. Furthermore, if $X = (T_X, v_X)$ and $t \not\in T_X$, then $X$ has no value at time $t$.
7.1 Operations on flows

Let \( X = (T_X, v_X) \) be a flow on \( V_X \), \( Y = (T_Y, v_Y) \) be a flow on \( V_Y \) and, for each \( i \), \( X_i = (T_{X_i}, v_{X_i}) \) be a flow on \( V_{X_i} \).

**Constants**

Let \( k \in V \) be a constant. Then \( k \) is a flow on \( V \) defined by:

\[
T_Z = \{ \text{start} - (0, 1) \}, \\
v_Z(t) = \{ <\text{start} - (0, 1), k> \}.
\]

The value \( k \) is available “before the beginning of time”.

**Data operations**

Let \( f : V_{X_1} \times \cdots \times V_{X_n} \rightarrow V_Z \) be a mapping. Then \( Z = f(X_1, \ldots, X_n) \) is a flow on \( V_Z \) defined by:

\[
T_Z = \bigcap_{i=1..n} T_{X_i}, \\
v_Z(t) = f(v_{X_1}(t), \ldots, v_{X_n}(t)).
\]

At any instant, an operation is only applied to its operands if each of the operands is available \emph{exactly} at that instant. Should one of the operands not be available, then no operation is made and the available operands disappear into cyberspace.

For the purposes of the relational algebra, the operations \( \cap, \cup, \times, -, \pi_X, \sigma_F, \text{sum}_x, \text{avg}_x, \text{count}, \text{max}_x \) and \( \text{min}_x \) are all considered mappings.

<table>
<thead>
<tr>
<th>Table 1: Data operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>a+b</td>
</tr>
</tbody>
</table>

**Default**

\( Z = X \text{ default } Y \) is the flow \((T_Z, v_Z)\) on \( V_X \cup V_Y \) defined by:

\[
T_Z = T_X \cup T_Y, \\
v_Z(t) = \begin{cases} 
  v_X(t) & \text{if } t \in T_X, \\
  v_Y(t) & \text{if } t \in T_Y - T_X.
\end{cases}
\]

If only one value arrives at time \( t \), then \( Z \) produces this value; if two values arrive together, then \( Z \) produces the \( X \).

<table>
<thead>
<tr>
<th>Table 2: Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>a default b</td>
</tr>
</tbody>
</table>
Memory

$Z = X \text{ before } Y$ is the flow $(T_Z, v_Z)$ on $V_X$ defined by:

$$T_Z = \begin{cases} \{ t \in T_Y \mid \min(T_X) < t \} & \text{if } T_X \neq \emptyset, \\ \emptyset & \text{otherwise,} \end{cases}$$

$$v_Z(t) = v_X(\sup\{ t' \in T_X \mid t' < t \}).$$

If $Y$ produces a value at time $t$, then $Z$ produces at time $t$ the last value produced by $X$ strictly before $t$. Should $X$ never have produced a value before $t$, there is no input.

<table>
<thead>
<tr>
<th>a</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>a before b</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Memory

<table>
<thead>
<tr>
<th>a</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>a before a</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Memory shift

Time-stamp filtering

$Z = X \mid Y$ is the flow $(T_Z, v_Z)$ on $V_X$ defined by:

$$T_Z = T_X \cap T_Y,$$

$$v_Z(t) = v_X(t).$$

The flow $Z$ produces those values of $X$ that are simultaneous with values of $Y$.

<table>
<thead>
<tr>
<th>a</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>a before a</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Time-stamp filtering

Value filtering

$Z = \varphi_F X$ is the flow $(T_Z, v_Z)$ on $V_X$ defined by:

$$T_Z = \{ t \in T_X \mid v_X(t) \in F \},$$

$$v_Z(t) = v_X(t).$$

If a value arrives at time $t$, then $Z$ produces it only if it is true, otherwise it is discarded.

8 Interpreting queries

Let $A$ be an atom of the form $p(e_1, \ldots, e_n)$, where $p$ is a predicate symbol and all the $e_i$ are terms. Formulas of the form $[t_0, t_1] \mathbf{present}_p$ and $[t_0, t_1] A$ are called canonical terms. The idea is that
\[ [t_0, t_1] A \text{ states that } A \text{ is satisfied from time } t_0 \text{ to } t_1. \] As for \([t_0, t_1]\) \text{ present}_p, \text{ it states that the predicate } p \text{ is present from time } t_0 \text{ to } t_1, \text{ even if there may be no terms } e_i \text{ for which } p(e_1, \ldots, e_n) \text{ is satisfied; } \text{ present}_p \text{ is necessary to distinguish instants in which no information about } p \text{ is available and instants where } p \text{ is present but never satisfied.}

The semantics of NSTL is based on the Herbrand universe. For a given program \( P \), the Herbrand universe of \( P \) is written \( U_P \), which should be finite. A non-standard temporal interpretation \( A \) of \( P \) assigns each predicate symbol used in \( P \) a partial mapping from the collection of instants to finite relations over \( U_P \). Let \( \text{Pred} \) be the set of all predicate symbols appearing in \( P \). Then

\[
A \in \left[ \text{Pred} \to \bigcup_{n \geq 0} \left[ T \to \mathcal{P}(U^n_P) \right] \right],
\]

where \([X \to Y]\) means the set of all total functions from \( X \) to \( Y \); \([X \rightarrow Y]\) the set of all partial functions from \( X \) to \( Y \); \( X^n \) the \( n \)-fold Cartesian product of \( X \) and \( \mathcal{P}(X) \) the powerset of \( X \).

Given an NSTL program \( db \) (a non-standard temporal deductive database), a NSTRA expression contains only those predicate symbols used in \( db \) and terms from the Herbrand universe of \( db \). Let \( E \) be an NSTRA relation: then \( [E][db] \) is the denotation of \( E \) with respect to \( db \). Therefore,

\[
[E] \in \left[ \text{DB} \to \bigcup_{n \geq 0} \left[ T \to \mathcal{P}(U^n_P) \right] \right],
\]

where \( \text{DB} \) is the set of NSTRA programs and \( U \) is the set of ground terms of the non-standard temporal logic. The definitions of the denotations of each kind of NSTRA expression are as follows.

\[
[p][db] = (\cap \mathcal{M}(db))(p) \text{ default } \left( \emptyset \text{ before } (\cap \mathcal{M}(db))(\text{present}_p) \right)
\]

\[
\nabla_1 E[db] = \nabla_1[E][db]
\]

\[
E \nabla_2 E'[db] = [E][db] \nabla_2 [E'][db]
\]

where \( p \) is a predicate symbol; \( \nabla_1 \) is a unary NSTRA operator (\( \varphi_F, \pi_X, \sigma_X, \text{sum}_x, \text{avg}_x, \text{count}, \text{max}_x \) and \( \text{min}_x \)); and \( \nabla_2 \) is a binary NSTRA operator (\text{default}, \text{before}, [\text{\_}], \cap, \cup, \times \) and \( (\cdot) \).

Since NSTRA works at the relation level, and NSTL works at the tuple level, there is no need for a direct equivalence between NSTRA expressions and NSTL programs. Nevertheless, NSTL has an expressive power so similar to NSTRA that the intensional definition of relations, i.e. NSTL programs, may be elaborated with the following guideline in mind:

<table>
<thead>
<tr>
<th>NSTRA</th>
<th>NSTL Horn clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varphi<a href="A">\text{true}</a> )</td>
<td>( A(\text{true}) )</td>
</tr>
<tr>
<td>( A \rightarrow B )</td>
<td>( A(\overline{x}), B ) the usual conjunction</td>
</tr>
<tr>
<td>( A \text{ default } B )</td>
<td>( C(\overline{x}) ::= A(\overline{x}) ). either ( A ) or</td>
</tr>
<tr>
<td></td>
<td>( C(\overline{x}) := \neg A(\overline{x}), B(\overline{x}) ). ( B ) is true</td>
</tr>
<tr>
<td>( A \text{ before } B )</td>
<td>( A(\overline{x}); B ) we are looking for instances of ( A )</td>
</tr>
</tbody>
</table>

Since the “default” operator is a switch, it must be translated into an intermediate predicate \( C \).

The “;” operator is a logical serial operator meaning that “\( A; B \)” is true at time \( t \) iff \( B \) is true at time \( t \) and \( A \) is true at a time \( t' < t \).

The implicit handling of \( \text{present}_p \) in NSTRA must be explicit in NSTL so the programmer must be able to deal with this situation.

Table 6: Value filtering

<table>
<thead>
<tr>
<th>a</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varphi_{(2,3)}a )</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Below, we present an example in which the choice of time domain has major impact on the semantics of the example. Consider the following three nstra equations:

\begin{align*}
\text{hasAppearedR} &= R \text{ before true}; \\
\text{hasAppearedQ} &= \text{hasAppearedR} \text{ before true}; \\
\text{on} &= \text{hasAppearedR} \text{ and not hasAppearedQ};
\end{align*}

which correspond to the three following nstl rules:

\begin{align*}
\text{hasAppearedR} &:- R ; \text{true}. \\
\text{hasAppearedQ} &:- \text{hasAppearedR} ; \text{true}. \\
\text{on} &:- \text{hasAppearedR}, \text{not hasAppearedQ}.
\end{align*}

If the time domain is discrete, then the equations ensure that on becomes true the instant following the first appearance of R. On the other hand, if we reduce the granularity to zero, then the program has no meaning, since the before operator is not well-defined over the real domain. However, using an infinitesimal granularity, as in this paper, allows us to give meaning to such programs without having to limit the granularity of databases. In this particular instance, the on relation is valid over one micro-instant, the one immediately following the first micro-instant at which R is valid.

9 Discussion

We have defined a temporal database system that supposes a non-standard time line. It is composed of a logic, nstl, essentially DATALOG with an additional operator, “;”, and of nstra, the relational algebra with three temporal operators.

Queries are made in nstra, which is capable of applying aggregate operations on results provided by the deductive database written in nstl. The temporal operations can be used to make queries about relations with different rates of validity.

The limitation of the current system is that the deductive system cannot make computations for the future. To do this would require some sort of delay operator; this problem is currently being looked at.

10 Acknowledgements

We would like to thank André Arnold, who came up with the basic primitives in Blizzard. We would also like to thank one of the referees, who forced us to come up with an intuitive explanation for non-standard time.

References


