J. Symbolic Computation (1996) **22**, 649–664

A Non-standard Temporal Deductive Database System

JEAN-RAYMOND GAGNÉ† AND JOHN PLAICE†

Département d'informatique, Université Laval,
Ste-Foy (Québec) Canada G1K 7P4

(Received 31 October 1995)

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 (NSTRA). 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 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 NSTRA 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) showed 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

† E-mail: gagne@ift.ulaval.ca, plaice@acm.org

0747–7171/96/110649 + 16 $25.00/0

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 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 $\langle r, z\rangle$, 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 Borgne *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 continously 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 formulae, 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.

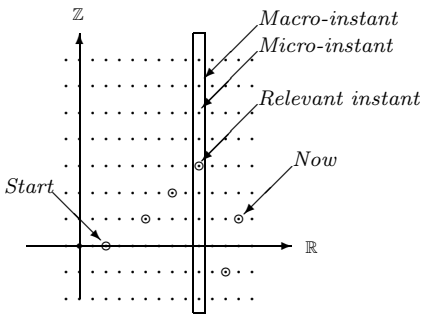

Figure 1. Time domain.

DEFINITION 2.1. *The time domain is the set* $\mathbb{T} = \mathbb{R} \times \mathbb{Z}$*, endowed with the natural lexicographic order. Each real* r *determines a* macro-instant *and each pair* $t = \langle r, z \rangle$*, where* $r \in \mathbb{R}$ *and* $z \in \mathbb{Z}$*, defines a* micro-instant*. The two components of* $\langle r, z \rangle$ *are accessed through projection functions* $\pi_{\mathbb{R}}$ *and* $\pi_{\mathbb{Z}}$*, such that* $r = \pi_{\mathbb{R}}\langle r, z \rangle$ *(the* real part*) and* $z = \pi_{\mathbb{Z}}\langle r, z \rangle$ *(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 2.2. *Let* $t_0 = \langle r_0, z_0 \rangle$ *and* $t_1 = \langle r_1, z_1 \rangle$ *be two micro-instants in* $\mathbb{T}$*. 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 2.3. *Let* t_0 *and* t_1 *be micro-instants in* $\mathbb{T}$*. Then* $[t_0, t_1]$ *defines an* interval *over* $\mathbb{T}$*: For all* $t \in \mathbb{T}$ *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* Δ*.*

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 2.4. *The names* start *and* now *refer, respectively, to the first and last micro-instants that are relevant to an application. The interval* $\mathbf{U}_{\mathbb{T}} = [\text{start}, \text{now}]$*, 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 formulae that are never true.

DEFINITION 2.5. *A* temporal element I *is a finite union of intervals in* $\mathbf{U}_{\mathbb{T}}$*. This set of intervals is closed under finite applications of union, intersection, and complementation. With* $\mathbf{U}_{\mathbb{T}}$ *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 2.6. *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 2.1. *Let* $I \subseteq \mathbf{U}_{\mathbb{T}}$*. 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 $\mathcal{V}$ of variables. We write $T_\Sigma[\mathcal{V}]$ for the set of terms (including variables) over Σ and At_Σ for the set of atoms over Σ.

DEFINITION 3.1. *Let* $p \in \Sigma$ *be a predicate symbol,* n *be the arity of* p *and* $x_1, \ldots, x_n$ *be terms in* $T_\Sigma[\mathcal{V}]$*. 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 3.2. *Let* Δ *be an interval in* $\mathbf{U}_{\mathbb{T}}$ *and* p *a simple atom. Then* Δp *is a* time-stamped atom*, and* Δ *is called a* timestamp*. The meaning of* Δp *is that* p *is valid during the interval* Δ*.*

DEFINITION 3.3. *An* NSTL atom *is either a simple atom or a timestamped atom. An arbitrary* NSTL *atom is written* α.

DEFINITION 3.4. *The set of* NSTL formulae *is given by the following grammar:*

$$\varphi ::= \alpha \mid \neg\alpha \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi;\varphi \mid (\forall X)\varphi$$

where φ *stands for an arbitrary* NSTL *formula.*

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

DEFINITION 3.5. *The set of* NSTL conjunctions *is given by the following grammar:*

$$\chi ::= \alpha \mid \neg\alpha \mid \chi \wedge \chi \mid \chi;\chi$$

where χ *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 3.6. *An* NSTL Horn rule *is a formula of the form* $\alpha \leftarrow \chi$, *where* α *is an* NSTL *atom and* χ *is an* NSTL *conjunction.*

DEFINITION 3.7. *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 Σ-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 Σ.

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

DEFINITION 4.1. *An* NSTL Σ-interpretation A *consists of:*

1. *a non-empty set of constants called* $\mathrm{dom}(A)$*;*
2. *an element* $c_A \in \mathrm{dom}(A)$ *for each constant* $c \in \Sigma$*;*
3. *a relation* $p_A \subseteq \mathrm{dom}(A)^n$ *for each n-ary predicate symbol* $p \in \Sigma$*;*
4. *a temporal element* $I_\alpha \subseteq \mathbf{U}_\mathbb{T}$ *for each* $\alpha \in \mathrm{At}_\Sigma$.

The definition of a Herbrand interpretation is standard.

DEFINITION 4.2. *An* NSTL Herbrand interpretation *is a* Σ*-interpretation for which* $\mathrm{dom}(A)$ *is the set of all ground atoms of* Σ.

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

DEFINITION 4.4. *An* NSTL Herbrand base *is a set of* NSTL *facts, one for each ground simple atom $\alpha \in \mathrm{At}_\Sigma$. This base states when each of these formulae is valid.*

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

PROPOSITION 4.1. *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. □

PROPOSITION 4.2. *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:

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

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 Σ by:

$$p_H = \{\vec{x} \mid Ip(\vec{x}) \in H_b\}$$

and for every atom $p(\vec{x}) \in \mathrm{At}_\Sigma$, $I_{p(\vec{x})}$ is equal to the temporal element associated to $p(\vec{x})$ in H. □

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

$$\begin{array}{lll}
H \models_t p(\vec{x}) & \textit{iff} & \exists\, Ip(\vec{x}) \in H \textit{ such that } t \in I, \\
H \models_t [t_0, t_1]\varphi & \textit{iff} & t_1 \leq t \textit{ and } (\forall t' \in [t_0, t_1])H \models_{t'} \varphi, \\
H \models_t \neg\varphi & \textit{iff} & H \not\models_t \varphi, \\
H \models_t (\varphi \wedge \psi) & \textit{iff} & H \models_t \varphi \textit{ and } H \models_t \psi, \\
H \models_t (\varphi \vee \psi) & \textit{iff} & H \models_t \varphi \textit{ or } H \models_t \psi, \\
H \models_t (\varphi; \psi) & \textit{iff} & H \models_t \psi \textit{ and } (\exists\, t' < t)H \models_{t'} \varphi, \\
H \models_t (\forall X)\varphi & \textit{iff} & \textit{for every } x \in T_\Sigma, H \models_t \varphi[X/x],
\end{array}$$

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

DEFINITION 4.6. *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 φ in P and for all $t \in \mathbf{U}_{\mathbb{T}}$, $H \models_t \varphi$.*

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 4.7. *The* actual part of an NSTL conjunction χ *is* $\Upsilon(\chi)$*, defined case by case as follows:*

$$\begin{aligned}
\Upsilon(p) &= \{p\}, \qquad p \text{ is an atom},\\
\Upsilon(\Delta p) &= \{p\},\\
\Upsilon(\neg p) &= \{\neg p\},\\
\Upsilon(\neg\Delta p) &= \{\neg p\},\\
\Upsilon(\chi_1 \wedge \chi_2) &= \Upsilon(\chi_1) \cup \Upsilon(\chi_2),\\
\Upsilon(\chi_1; \chi_2) &= \Upsilon(\chi_2).
\end{aligned}$$

DEFINITION 4.8. *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 4.9. *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 *$\mathcal{D}(P)$, 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 4.10. *Let p and q be ground* NSTL *simple atoms. Then $p \overset{+}{\preceq} q$ iff there is a directed path from q to p.*

DEFINITION 4.11. *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 4.12. *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 4.13. *The expression $M \ll M'$ holds iff for any atom α and any micro-instant t, if $M \models_t \alpha$ and $M' \not\models_t \alpha$, then there exists β such that $\beta \preceq \alpha$, $M \not\models_t \beta$ and $M' \models_t \beta$.*

DEFINITION 4.14. *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 5.1. (TARSKI–KNASTER) *Let G be a monotone and continuous function. Then*

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

The μ 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 T_P operator used for the semantics of logic programs is modified to take into account the specific micro-instant.

DEFINITION 5.1. *Let P be an* NSTL *program, t be a micro-instant and H be an* NSTL *Herbrand base.*

$$T_{P,t,\underline{H}}(H) = \underline{H} \cup \{\theta\alpha \mid \alpha \leftarrow \chi \in P \text{ and } \theta : \mathcal{V} \to T_\Sigma \text{ and } (H \cup \underline{H}) \models_t \theta\chi\}.$$

DEFINITION 5.2. *The fixpoint for a micro-instant t is $\mu T_{P,t,\underline{H}}$, where $\underline{H}$ 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 Λ operator.

DEFINITION 5.3. *The set of* potentially relevant micro-instants *for an* NSTL *rule* φ *is* $\Lambda_H(\varphi)$.

$$\begin{aligned}
\Lambda_H(p) &= \cup\{\Lambda(I) \mid \exists\theta : \mathcal{V} \to T_\Sigma \text{ and } Ip\theta \in H\},\\
\Lambda_H([t_0,t_1]p) &= \{t_1\},\\
\Lambda_H(\neg\varphi) &= \Lambda_H(\varphi),\\
\Lambda_H(\varphi \wedge \psi) &= \Lambda_H(\varphi) \cup \Lambda_H(\psi),\\
\Lambda_H(\varphi;\psi) &= \Lambda_H(\psi) \cup \{t + \langle 0,1\rangle \mid t \in \Lambda_H(\varphi)\},\\
\Lambda_H(p \leftarrow \chi) &= \Lambda_H(\chi),\\
\Lambda_H([t_0,t_1]p \leftarrow \chi) &= \{t_0,t_1\} \cup [t_0,t_1] \cap \Lambda_H(\chi)
\end{aligned}$$

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

DEFINITION 5.4. *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 5.5. *An* NSTL *program successively applies, in order, the fixpoint calculations for all relevant micro-instants:*

$$F_P = \bigcup_{i=0,\ldots,|M|} F_{P,i}(\emptyset),$$

where

$$\begin{aligned}
F_{P,0}(H) &= H,\\
F_{P,i}(H) &= \mu T_{P,M_i,(F_{P,i-1}(H))},\\
M &= \{\text{start}\} \cup [\text{start},\text{now}] \cap \Lambda_P(\{\mathbf{U}_\mathbb{T}\} \times \text{At}_\Sigma).
\end{aligned}$$

This version of the function F computes a fixpoint for a micro-instant and then advances to the the next *relevant* micro-instant. This process begins with the first relevevant 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 6.1. *Let P be a* NSTL *program for which the directed graph $\mathcal{D}(P)$ 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:*

$$L_P = \{Ip(x_1, \ldots, x_n) \mid x_1, \ldots, x_n \in T_\Sigma \text{ and } I \subseteq \mathbf{U}_\mathbb{T} \text{ and } (\forall t \in I) P \models_t p(x_1, \ldots, x_n)\}.$$

PROOF. Let $\alpha \leftarrow \beta_1 \wedge \ldots; \ldots \wedge \beta_i \wedge \ldots; \ldots \wedge \beta_m$ 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

$$\alpha \leftarrow \beta_1^{k_1} \ldots \beta_m^{k_m},$$

where each k_i denotes the number of ";" to the right of the β_i.

So, let $X_1, \ldots, X_n$ be the variables in α and the β_i. Then L_P is a perfect model of P can be written as follows:

$$L_P \models \alpha \leftarrow \beta_1^{k_1} \ldots \beta_m^{k_m}.$$

This holds if for all $t \in \mathbf{U}_\mathbb{T}$,

$$L_P \models_t \forall X_1 \ldots \forall X_n (\beta_1^{k_1} \ldots \beta_m^{k_1} \rightarrow \alpha).$$

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

$$t_{k1} < t_{k2} < \cdots < t_{ki} < \cdots < t_{km} = t,$$

where each $t_{ki} \in \mathbf{U}_\mathbb{T}$, and for all ground terms $x_1, \ldots, x_n$, then

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

This last implication can be translated into

$$\begin{aligned}&(\exists I_{\beta_i}) I_{\beta_i} \beta_i[X_1/x_1, \ldots, X_n/x_n] \in L_P \text{ and } t_{ki} \in I_{\beta_i}, i = 1, \ldots, m\\ &\Rightarrow (\exists I_\alpha) I_\alpha \alpha[X_1/x_1, \ldots, X_n/x_n] \in L_P \text{ and } t \in I_\alpha;\end{aligned}$$

which holds if

$$P \models_{t_{ki}} \beta_i^{k_i}[X_1/x_1, \ldots, X_n/x_n], i = 1, \ldots, m \Rightarrow P \models_t \alpha[X_1/x_1, \ldots, X_n/x_n];$$

which is true, since $P \models_t ((\beta_1 \wedge \ldots; \ldots \wedge \beta_i \wedge \ldots; \ldots \wedge \beta_m) \rightarrow \alpha)[X_1/x_1, \ldots, X_n/x_n]$.

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 formulae that follow from P, it must also be a model of all formulae in H_{L_P}. Since formulae 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 $\sqcap\mathcal{M}(P) = L_P$. □

THEOREM 6.2. *The set $\sqcap\mathcal{M}(P)$ is computable and equal to F_P.*

PROOF. We first prove that $\sqcap\mathcal{M}(P)$ 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 $F_{p,i-1}(H)$, $i \geq 1$. We wish to prove that $F_{p,n}(H)$ is also a fixpoint:

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

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

$$F_P = L_P = \sqcap\mathcal{M}(P).$$

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 $\mu_X T_{P,t}(H)$ is computable, it follows that F_P is computable. □

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 $\mathbf{U}_\mathbb{T} = [start, now]$ be the time domain as defined in Section 2. A subset T of $\mathbf{U}_\mathbb{T}$ is called a date set. A flow X on V dated by $\mathbf{U}_\mathbb{T}$ is a pair (T_X, v_X), where T_X is a date set and $v_X : T_X \to V$.

It is important to note that a flow can be finite, even empty. Furthermore, if $X = (T_X, v_X)$ and $t \notin T_X$, 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}.

Table 1. Data operations.

a	=	1	2	4	3		2
b	=	3		5		6	7
a + b	=	4		9			9

Table 2. Default.

a	=	1	2	3		2
b	=	3			6	7
a `default` b	=	1	2	3	6	2

CONSTANTS

Let $\mathbf{k} \in V$ be a constant. Then k is a flow on V defined by:

$$T_Z = \{start - \langle 0, 1\rangle\},$$
$$v_Z(t) = \mathbf{k}.$$

The value $\mathbf{k}$ is available "before the beginning of time".

DATA OPERATIONS

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

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

At any instant, an operation is only applied to its operands if each of the operands is available *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 $\cap$, $\cup$, $\times$, $-$, π_X, σ_F, $\mathtt{sum}_{\mathtt{x}}$, $\mathtt{avg}_{\mathtt{x}}$, `count`, $\mathtt{max}_{\mathtt{x}}$ and $\mathtt{min}_{\mathtt{x}}$ are all considered mappings.

DEFAULT

$Z = X$ `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 3. Memory.

a	=	1	2	4	3		2
b	=	3		5		6	7
a `before` b	=			2		3	3

Table 4. Memory shift.

a	=	1	2	4	3		2
b	=	3		5		6	7
a `before` a	=		1	2	4		3

MEMORY

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

$$T_Z = \begin{cases} \{t \in T_Y \mid \min(T_X) < t\} & \text{if } T_X \neq \emptyset, \\ \emptyset & \text{otherwise,} \end{cases}$$
$$v_Z(t) = v_X\big(\sup\{t' \in T_X \mid t' < t\}\big).$$

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.

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.

VALUE FILTERING

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

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

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

Table 5. Time-stamp filtering.

a	=	1	2	4	3		2
b	=	3		5		6	7
a | b	=	1		4			2

Table 6. Value filtering.

a	=	1	2	4	3	2
$\varphi_{\{2,3\}}$a	=		2		3	2

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 *Pred* be the set of all predicate symbols appearing in P. Then

$$A \in \Big[\mathit{Pred} \to \bigcup_{n\geq 0} \big[\mathbb{T} \rightharpoonup \mathcal{P}(U_P^n)\big]\Big],$$

where $[X \to Y]$ means the set of all total functions from X to Y; $[X \rightharpoonup Y]$ 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,

$$[\![E]\!] \in \Big[\mathcal{DB} \to \bigcup_{n\geq 0} \big[\mathbb{T} \rightharpoonup \mathcal{P}(U_P^n)\big]\Big],$$

where $\mathcal{DB}$ is the set of NSTL 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.

$$\begin{aligned}
[\![p]\!](db) &= \big(\sqcap\mathcal{M}(db)\big)(p)\ \texttt{default}\ \big(\emptyset\ \texttt{before}\ \big(\big(\sqcap\mathcal{M}(db)\big)(\texttt{present}_p)\big)\big)\\
[\![\bigtriangledown_1 E]\!](db) &= \bigtriangledown_1 [\![E]\!](db)\\
[\![E \bigtriangledown_2 E']\!](db) &= [\![E]\!](db) \bigtriangledown_2 [\![E']\!](db)
\end{aligned}$$

where p is a predicate symbol; $\bigtriangledown_1$ is a unary NSTRA operator (φ_F, π_X, σ_F, $\texttt{sum}_\texttt{x}$, $\texttt{avg}_\texttt{x}$, $\texttt{count}$, $\texttt{max}_\texttt{x}$ and $\texttt{min}_\texttt{x}$); and $\bigtriangledown_2$ is a binary NSTRA operator (**default**, **before**, $|$, $\cap$, $\cup$, $\times$ and $(-)$).

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:

NSTRA	NSTL Horn clauses	
$\varphi_{\{\texttt{true}\}}(A)$	$A(\mathit{true})$	$A(\mathit{false})$ may be needed elsewhere
$A \mid B$	$A(\vec{x}), B$	the usual conjunction
A `default` B	$C(\vec{x}) :- A(\vec{x}).$	either A or
	$C(\vec{x}) :- \neg A, B(\vec{x}).$	B is true
A `before` B	$A(\vec{x}); B$	we are looking for instances of A

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 $\texttt{present}_p$ in NSTRA must be explicit in NSTL so the programmer must be able to deal with this situation.

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:

```
hasAppearedR = R before true;
hasAppearedQ = hasAppearedR before true;
on           = hasAppearedR and not hasAppearedQ;
```

which correspond to the three following NSTL rules:

```
hasAppearedR :- R ; true.
hasAppearedQ :- hasAppearedR ; true.
on           :- hasAppearedR, not hasAppearedQ.
```

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.

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

Boussinot, P., De Simone, R. (1991). The ESTEREL language. *Proc. IEEE*, **79**(9), 1293–1304.

Gadia, S.K. (1988). A homogeneous relational model and query languages for temporal databases. *ACM Transactions on Database Systems*, **13**(4), 418–448.

Gagné, J.-R., Plaice, J. (1995). A non-standard temporal deductive database system. In *Executable Temporal Logics: Working Notes*. Workshop of IJCAI 1995. Los Altos, CA: AAAI.

Halbwachs, N., Caspi, P., Raymond, P., Pilaud, D. (1991). The synchronous data flow programming language LUSTRE. *Proc. IEEE*, **79**(9), 1305–1320.

Iwasaki, Y., Farquhar, A., Saraswat, V., Bobrow, D., Gupta, V. (1995). Modeling time in hybrid systems: How fast is "instantaneous"? In *Proc. IJCAI 1995*, pp. 1773–1780. Los Altos, CA: AAAI.

Lloyd, F.W. (1987). *Foundations of Logic Programming*, 2nd edition. Berlin: Springer-Verlag.

McLauglin, W.I., Miller, S.L. (1992). An epistemological use of nonstandard analysis to answer Zeno's objections against motion. *Synthese*, **92**, 371–384.

Nelson, E. (1977). Internal set theory: A new approach to nonstandard analysis. *Bulletin of the American Mathematical Society*, **83**(7), 1165–1198.

Orgun, M.A., Wadge, W.W. (1992). A relational algebra as a query language for Temporal Datalog, In *Proc. DEXA'92, Database and Expert Systems Appplications*, pp. 276–281. Berlin: Springer-Verlag.

Le Borgne, M., Le Guernic, P., Gautier, T., Le Maire, C. (1991). Programming real-time applications with SIGNAL. *Proc. IEEE*, **79**(9), 1321–1336.

Przymusinski, T.C. (1988). On the declarative semantics of deductive databases and logic programs. In Minker, J., (ed.), *Deductive Databases and Logic Programming*, pp. 193–216. Los Altos, CA: Morgan Kaufmann.

Robinson, A. (1974). *Non-standard analysis*, revised edition. Amsterdam: North-Holland.

Tarski, A. (1955). A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, **5**, 285–309.

Ullman, J.D. (1988). *Principles of Database and Knowledge-Base Systems*, volume 1. Rockville, MD: Computer Science Press.

Wiederhold, G., Jajodia, S., Litwin, W. (1991). Dealing with granularity of time in temporal databases. In Andersen, R., Bubenko, J.A. jr, Sølvberg, A., (eds), *Advanced Information Systems Engineering*, pp. 124–140. Berlin: Springer-Verlag, LNCS series.