The synchronous approach to programming reactive systems supposes that the reaction to an input event is instantaneous, i.e. it takes no time. This approach, only realistic if the minimum delay between two (discrete) input events is greater than the reaction time, greatly simplifies the semantics and the programming of reactive systems. Should an a posteriori verification of generated code ensure that all real-time constraints are respected, then the synchrony hypothesis is a valid abstraction.

Once the synchronous model of computation is accepted, it is natural to think in terms of instants. An input even occurs, a reaction takes place, possibly by provoking an output event, all “in the same instant”. Because of the synchronous model, the input and output events are perceived to be simultaneous.

However, even if they take no time, actions taking place within an instant must remain ordered. This point of “infinitely fast but ordered” yields naturally a notion of macro-instants, each made up of a number of micro-instants. More formally, the time line is $T = \mathbb{R} \times \mathbb{N}$, with each real-numbered macro-instant $r$ associated with a countable set of micro-instants $(r, n)$.

A more general framework comes from non-standard analysis. According to Robinson, a non-standard real-number $*r \in *\mathbb{R}$ can be written

$$*r = \sum_{i=-\infty}^{\infty} \epsilon^i r_i, \quad r_i \in \mathbb{R},$$

where $\epsilon$ is a fixed infinitesimal value such that $\epsilon > 0$ and $\forall r \in \mathbb{R}, r > 0 \rightarrow \epsilon < r$.

If $\epsilon > 0$ is an infinitesimal, then $\epsilon^2$ is also infinitesimal, with $\forall r \in \mathbb{R}, r > 0 \rightarrow r\epsilon^2 < \epsilon$. In general, if $i > j$, then $\epsilon^i$ is infinitely smaller than $\epsilon^j$:

$$\forall i, j \in \mathbb{Z}, i > j \rightarrow (\forall r \in \mathbb{R}, r > 0 \rightarrow r\epsilon^i < \epsilon^j).$$
If $i < 0$, then $\epsilon^i$ is an infinite value. In the following discussion, infinite values will not normally be used, which means that $*r$ can then be written as

$$*r = \sum_{i=0}^{\infty} \epsilon^i r_i.$$  

Infinitesimals were first used by Leibniz in the seventeenth century when he introduced the infinitesimal calculus. Euler used them enthusiastically in the eighteenth century, but they were dropped when Cauchy introduced the $\epsilon$-$\delta$ approach to limits. It was not until 1963 that Robinson gave a firm logical foundation for the infinitesimals and the non-standard reals. See Robinson’s book *Non-standard Analysis* for a full discussion.

In the non-standard approach to time, $T = *R$, i.e. the timestamps are non-standard reals. The set $T$ is partitioned into levels. A non-standard real

$$*r = \sum_{i=0}^{\infty} \epsilon^i r_i$$

is called an $n$-th level timestamp if $r_n \in \mathbb{Z}$ and $\forall i > n, r_i = 0$. Hence, 4 is 0-th level, $4.2 + 3\epsilon$ is first level and $1.2 - 2.3\epsilon^2 + 5\epsilon^3$ is third level. The intuition for $r_n \in \mathbb{Z}$ is that at some level, time can be discretized, so successive timestamps can be designated for describing discrete changes. The set of $n$-th level timestamps is written $T_n$.

Of course, $n$-level timestamps can be written as $(n+1)$-tuples. The three examples above become $(4), (4.2, 3)$ and $(1.2, 0, -2.3, 5)$. As for the micro-instants from the top of the page, they are first-level timestamps.

The authors have defined a new synchronous language, Blizzard, defined over the time line $T = *R$. Key to the design of Blizzard is that at some level, every timestamp has a unique previous and a unique next instant.

Let $D$ be a set of values. A timed dataflow $x$ defined over $D$ is a pair $(C_x, V_x)$, where $C_x \subseteq T$ is the set of timestamps for which $x$ is defined, and $V_x : C_x \to D$ is a function defined for each of the timestamps. This function is, of course, also a partial function $T \to D$.

Now the Blizzard language can be defined. Let $x = (C_x, V_x)$ be a flow on $D_x$, $y = (C_y, V_y)$ be a flow on $D_y$, and, for each $i$, $x = (C_x, V_x)$ be a flow on $D_{x_i}$. Let $k$ be a constant and let $f : D_{x_1} \times \cdots \times D_{x_n} \to D_z$ be a memoryless data operation. The semantics of the Blizzard operators are given in Table 1.
Table 1: Semantics of Blizzard operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Semantics</th>
</tr>
</thead>
</table>
| $z = k_n$ | $\mathcal{C}_z = T_n$  
$\mathcal{V}_z t = k$ |
| $z = f(x_1, \ldots, x_n)$ | $\mathcal{C}_z = \bigcap_{i=1..n} \mathcal{C}_{x_i}$  
$\mathcal{V}_z t = f(\mathcal{V}_{x_1} t, \ldots, \mathcal{V}_{x_n} t)$ |
| $z = x \nabla y$ | $\mathcal{C}_z = \mathcal{C}_x \cup \mathcal{C}_y$  
$\mathcal{V}_z t = \begin{cases} \mathcal{V}_x t & \text{if } t \in \mathcal{C}_x \\ \mathcal{V}_y t & \text{otherwise} \end{cases}$ |
| $z = \delta_n x$ | $\mathcal{C}_z = \{ t \in T_n \mid \exists t' < t : t' \in \mathcal{C}_{x} \}$  
$\mathcal{V}_z t = \mathcal{V}_z (\max \{ t' \in \mathcal{C}_x \mid t' < t \})$ |
| $z = x \mid y$ | $\mathcal{C}_z = \mathcal{C}_x \cap \mathcal{C}_y$  
$\mathcal{V}_z t = \mathcal{V}_y t$ |
| $z = \sigma x$ | $\mathcal{C}_z = \{ t \in \mathcal{C}_x \mid \mathcal{V}_x t = \text{true} \}$  
$\mathcal{V}_z t = \text{true}$ |