International Journal of Computers and Applications, Vol. 25, No. 2, 2003

REAL-TIME COMPUTATION: A FORMAL DEFINITION AND ITS APPLICATIONS

S.D. Bruda* and S.G. Akl*

Abstract

The concept of real time has different meanings in the systems and theory communities. Thus, the existing formal real-time models do not capture all the practically relevant aspects of such computations. This article proposes a new definition that, we believe, allows a unified treatment of all practically meaningful variants of real-time computations. We use the developed formalism to mode two important features of real-time algorithms, namely the presence of deadlines and the real-time arrival of input data. We also emphasize the expressive power of our model by using it to formalize aspects from the areas of real-time database systems and *ad hoc* networks. We offer formulations of the recognition problem for real-time database systems and of the routing problem in *ad hoc* networks. Finally, we suggest a variant of our formalism that is suited for modelling parallel distributed real-time algorithms. We believe that the proposed formalism is a first step towards a unified and realistic complexity theory for real-time parallel computations.

Key Words

Real-time computation, timed languages, ω-languages, parallel complexity, real-time databases, *ad hoc* networks

1. Introduction

The area of real-time computations has a strong practical grounding in domains-like operating systems, databases, and the control of physical processes. Besides these practical applications, however, research in this area is primarily focused on formal methods and on communication issues in distributed real-time systems. By contrast, little work has been done in the direction of applied complexity theory. In fact, the limited extent of this work is emphasized by the fact that even a realistic general definition for real-time algorithms is missing, although implicit definitions can be found in many places. Some papers have tried to address this issue, providing abstract machines that model real-time algorithms. In this context, the real-time Turing machine [1, 2] fails to capture many aspects that are important in practice, whereas the real-time producer/consumer paradigm [3] is suitable for modelling certain real-time systems rather than for developing a general complexity theory. Finally, the power of the the languages recognized by timed automata [4] is limited, as there are real-time problems that cannot be formalized as languages recognizable by memoryless finite-state models.

Indeed, the domain of real-time systems is very complex, with requirements varying from application to application. This complexity is probably the main obstacle to a unified theory. In this work, we try to address this issue. We believe that the model of timed languages [4] is a powerful tool, but the device used as acceptor is rather weak. We suggest, therefore, an extension of this study. We keep most of the important ingredients in the definition of timed languages, but we apply such a definition to a larger extent, suggesting a general model for the acceptors of such languages. We introduce a variant of the model from [4], called well-behaved timed ω-languages. Then we present the structure of an acceptor for them. We believe that our construction captures all the practical aspects of real-time computations. That is, our thesis is that well-behaved timed ω-languages model exactly all real-time computations.

Starting from Section 3, this article can be viewed as split into two main parts. The first part (Section 3) introduces the theory of timed ω-languages and real-time algorithms. The second part (Sections 4 and 5) supports our thesis. Indeed, we show in Section 4 how those ingredients that, when present, give to some problem the "real-time" qualifier (namely, computing with deadlines, and input data that arrive in real-time) can be modelled using our formalism. The expressiveness of the formalism is emphasized in Section 5, by modelling important practical problems from the domains of real-time database systems and *ad hoc* networks. We also identify in Section 6 a variant of our model that is particularly suited for studying parallel and distributed real-time systems by explicitly modelling the parallelism and distributedness of the system.

We believe that, starting from the formalism developed in this article, a unified complexity theory for real-time systems can be naturally developed.

* Department of Computing and Information Science, Queen's University, Kingston, Ontario, K7L 3N6 Canada; e-mail: {bruda, akl}@cs.queensu.ca

(paper no. 202-1217)

2. Preliminaries

Σ^* is the set of all the words of finite length over Σ; $\omega = |\mathbb{N}|$; Σ^ω contains exactly all the words over Σ of length ω. Given two (infinite or finite) words $\sigma = \sigma_1\sigma_2\cdots$ and $\sigma' = \sigma'_1\sigma'_2\cdots$, we say that σ' is a subsequence of σ iff (1) for each σ'_i there exists a σ_j such that $\sigma'_i = \sigma_j$, and (2) for any $i, j, k, l \geq 0$ such that $\sigma'_i = \sigma_j$ and $\sigma'_k = \sigma_l$, it holds that $i > k$ iff $j > l$. A finite automaton is a tuple $A = (\Sigma, S, s_0, \delta, F)$. Σ is the input alphabet, S is a (finite) set of states, s_0 is the initial state, $\delta \in S \times S \times \Sigma$ is the transition relation, and $F \subseteq S$ is the set of accepting states.

As our model is based on the theory of timed automata, it what follows we briefly review in what follows this theory, according to [4]. A Büchi-automaton is a finite-state automaton $A = (\Sigma, S, s_0, \delta, F)$ whose accepting condition is modified: given an (infinite) word $\sigma = \sigma_1\sigma_2\cdots$, the sequence $r = s_0 \stackrel{\sigma_1}{\rightarrow} s_1 \stackrel{\sigma_2}{\rightarrow} s_2 \stackrel{\sigma_3}{\rightarrow} \cdots$ is called a run of A over σ whenever $(s_{i-1}, s_i, \sigma_i) \in \delta$, $i > 0$. $\inf(r)$ is the set of all the states s in r such that $s = s_i$ for infinitely many i. σ is accepted iff there exists a run r such that $\inf(r) \cap F \neq \emptyset$.

A time sequence $\tau = \tau_1\tau_2\cdots$ is an infinite sequence of positive real values, such that the following constraints are satisfied: (1) monotonicity: $\tau_i \leq \tau_{i+1}$ for all $i \geq 0$; and (2) progress: for any $t \in \mathbb{R}$, there exists some $i \geq 1$ such that $\tau_i > t$. A timed ω-word over some alphabet Σ is a pair (σ, τ), where $\sigma \in \Sigma^\omega$ and τ is a time sequence. The time value τ_i associated with some symbol σ_i can be considered the time at which the corresponding symbol becomes available. A timed ω-language is a set of timed ω-words.

A clock x is a variable over $\mathbb{R}$, with two associated operations: reading the value of x and resetting x to zero. The value of x corresponds to the time elapsed from the moment when x has been most recently reset. For a set X of clocks, a set of constraints $\Phi(X)$ is defined by: $d \in \Phi(X)$ iff d is of the form $x \leq c$, $c \leq x$, $\neg d_1$, or $d_1 \wedge d_2$, for a constant c, $x \in X$ and $d_1, d_2 \in \Phi(X)$. A timed Büchi automaton (TBA) is a tuple $A = (\Sigma, S, s_0, \delta, C, F)$, where C is a finite set of clocks and $\delta \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C)$. If $(s, s', a, l, d) \in \delta$, then l is the set of clocks to be reset during the transition and $d \in \Phi(C)$. The transition is enabled only if d is valued to true using the current values of the clocks in C.

A run r of a TBA $A = (\Sigma, S, s_0, \delta, C, F)$ over (σ, τ) is an infinite sequence of the form: $r = (s_0, \nu_0) \stackrel{\sigma_1,\tau_1}{\longrightarrow} (s_1, \nu_1) \stackrel{\sigma_2,\tau_2}{\longrightarrow} (s_2, \nu_2) \stackrel{\sigma_3,\tau_3}{\longrightarrow} \cdots$ where $\sigma = \sigma_1\sigma_2\cdots$, $\tau = \tau_1\tau_2\cdots$, $\nu_i \in \{f | f\colon C \rightarrow \mathbb{R}\}$, $i \geq 0$, and the following conditions hold: $\nu_0(x) = 0$ for all $x \in C$; and for all $i > 0$, there exists a transition $(s_{i-1}, s_i, \sigma_i, l_i, d_i) \in \delta$ such that $(\nu_{i-1} + \tau_i - \tau_{i-1})$ satisfies d_i, for all $x \in C - l_i$, $\nu_i(x) = \nu_{i-1}(x) + \tau_i - \tau_{i-1}$, and, for all $x' \in l_i$, $\nu_i(x') = 0$. A timed ω-language accepted by some TBA is ω-regular.

3. Timed Languages

Even if timed regular languages look well suited for modelling real-time computations, the TBA is not sufficiently powerful for this purpose:

Theorem 1. There exist languages formed by infinite words (ω-languages) that are not ω-regular. Thus, there exist timed ω-languages that are not (timed) ω-regular.

Proof. Consider the (nonregular) language $L = \{a^u b^x c^v d^x | u, x, v > 0\}$. Let $L_\omega = \{l_1\$l_2\$l_3\$\cdots | l_i \in L, i > 0, \$ \notin \Sigma\}$. Assume that L_ω is ω-regular. Then there exists a run r of some Büchi automaton A over $x \in L_\omega$ such that $\inf(r) \cap F \neq \emptyset$. Let S_1 (S_2) be the set of all the states in r that A is into immediately after (before) parsing a symbol \$, $S_1, S_2 \subseteq S$. We construct a finite automaton A' that recognizes L: the initial state of is $s' \notin S$; the set of states is $S \cup \{s'\}$; the set of final states is S_2; and the transition relation is δ, augmented with λ-transitions from s' to each state in S_1. The existence of A' is a contradiction. □

Note that the language L_ω built in the proof of Theorem 1 is not uninteresting from a practical point of view, as it models a search into a database ($a^u b^x c^v$) for a given key (d^x) matched in the database by b^x. We just found some practical situation that does not pertain to the class of (timed) ω-regular languages.

3.1 A Formal Definition

Despite the limited scope of the finite-state approach, the concept of timed languages is a very powerful one. We therefore propose a definition that is similar to the one in [4], but is not restricted to finite-state acceptors. Our presentation is clearer if we use a slightly modified concept of time sequence.

Definition 1. A sequence $\tau \in \mathbb{N}^\omega$, $\tau = \tau_1\tau_2\cdots$, is a time sequence if it is an infinite sequence of positive values, such that the monotonicity constraint is satisfied. A finite subsequence of a time sequence is also a time sequence. A well-behaved time sequence is a time sequence for which the progress condition holds.

A time sequence may be finite or infinite, whereas a well-behaved time sequence is always infinite. In fact, a well-behaved time sequence, in our terminology, is identical to the concept of time sequence used in [4].

Definition 2. A (well-behaved) timed ω-word over an alphabet Σ is a pair (σ, τ), where τ is a (well-behaved) time sequence and, if $\tau \in \mathbb{N}^k$, then $\sigma \in \Sigma^k$, $k \in \mathbb{N} \cup \{\omega\}$. Given a symbol σ_i from σ, the associated τ_i of the time sequence τ represents the time at which σ_i becomes available as input. A (well-behaved) timed ω-language over some alphabet Σ is a set of (well-behaved) timed ω-words over Σ.

Definition 2 is an natural extension of the definition of timed regular languages, except that we added the "well-behaved" qualifier, generated by the modified terminology presented in Definition 1.

The union, intersection, and complement for timed ω-languages are straightforwardly defined. One can rely

on the semantics of timed words in defining a meaningful concatenation operation. Thus, the concatenation of two timed words is defined as the union of their sequences of symbols, ordered in nondecreasing order of their arrival time:[1] Let (σ', τ') and (σ'', τ'') be two timed ω-words. Then (σ, τ) is the concatenation of (σ', τ') and (σ'', τ'') ($(\sigma, \tau) = (\sigma', \tau')(\sigma'', \tau'')$) iff (1) τ is a time sequence; $(\sigma'_1, \tau'_1)(\sigma'_2, \tau'_2)\cdots$ and $(\sigma''_1, \tau''_1)(\sigma''_2, \tau''_2)\cdots$ are subsequences of $(\sigma_1, \tau_1)(\sigma_2, \tau_2)\cdots$; for any $i > 0$, there exists $j > 0$ and $d \in \{', ''\}$ such that $(\sigma_i, \tau_i) = (\sigma^d_j, \tau^d_j)$; (2) for any $d \in \{', ''\}$ and any $0 \leq i < j$, such that $\tau^d_k = \tau^d_l$ for any k, l, $i \leq k < l \leq j$, there exists m such that, for any $0 \leq \iota \leq j - i$, $(\sigma_{m+\iota}, \tau_{m+\iota}) = (\sigma^d_{i+\iota}, \tau^d_{i+\iota})$; and (3) for any $i, j \geq 0$ such that $\tau'_i = \tau''_j$, there exist k and l, $k < l$, such that $(\sigma_k, \tau_k) = (\sigma'_i, \tau'_i)$ and $(\sigma_l, \tau_l) = (\sigma''_j, \tau''_j)$. The concatenation of timed ω-words is clearly associative. Given two timed ω-languages L_1 and L_2, the concatenation of L_1 and L_2 is $L_1L_2 = \{w_1w_2 | w_1 \in L_1, w_2 \in L_2\}$. The notation $\prod_{i=1}^{n} w_i$ ($\prod_{i=1}^{n} L_i$) is a shorthand for $w_1w_2\cdots w_n$ ($L_1L_2\cdots L_n$). For some timed ω-language L, let $L^0 = \emptyset$, $L^1 = L$, and $L^k = LL^{k-1}$, $k > 1$. The Kleene closure of L is $L^* = \bigcup_{0 \leq k < \omega} L^k$.

Theorem 2. The set of (well-behaved) timed ω-languages is closed under intersection, union, complement, concatenation, and Kleene closure, under a proper definition of the latter two operations. A subset of a (well-behaved) timed ω-language is a (well-behaved) timed ω-language.

3.2 Accepting Timed Languages

In light of Definition 2, we can also establish the general form of an acceptor for timed languages:

Definition 3. A real-time algorithm A consists in a finite control (i.e., a program), an input tape (i.e., an input stream) that contains a timed ω-word, and a write-only output tape (i.e., an output stream) containing symbols from some alphabet Δ that are written by A. The input tape has the same semantics as a timed ω-word: if (σ_i, τ_i) is an element of the input tape, then σ_i is available for A at precisely the time τ_i. During any time unit, A may add at most one symbol to the output tape. $o(A, w)$ denotes the content of the output tape of A working on input w. A may have access to an infinite amount of storage space (working tape(s), RAM memory, etc.) outside the input and output tapes, but only a finite amount of this space can be used for any computation.

A real-time algorithm A accepts the timed ω-language L if, on any input w, $|o(A, w)|_f = \omega$ iff $w \in L$, for some designated symbol $f \in \Delta$.

[1] Intuitively, such an operation is similar to merging two sequences of pairs (symbol, time value), that are sorted with respect to the time values. Two additional constraints (ordering the result in the absence of any ordering based on the arrival time) are imposed, so that nondeterminism is eliminated.

It is assumed that the input of a real-time algorithm is always a (not necessarily well-formed) timed ω-word.

It is worth mentioning that the actual meaning of the symbol f might be different from algorithm to algorithm, but such a distinction is immaterial for the global theory of timed ω-languages. Indeed, consider an aperiodic real-time computation, for example, a computation with some deadline. If for some particular input, the computation meets its deadline, then from now on the real-time algorithm that accepts the language modelling this problem may keep writing f on the output tape; the first appearance of f signals a successful computation, but the subsequent occurrences do not add any information, being present for the sole purpose of respecting the acceptance condition. On the other hand, consider the timed language associated with a periodic computation, for example, a periodic query in a real-time database system. Then f might appear on the output tape each time an occurrence of the query is successfully served (obviously, a failure could prevent further occurrences of f, should the specification of the problem require that all the queries be served). In this case, each occurrence of f signals a successfully served query. However, even if the actual meaning of the fs on the output tape can vary from application to application, the acceptance condition remains invariant throughout the domain of real-time computations.

Note that a TBA is equipped with a set of clocks, as a finite automaton does not have access to any amount of storage space, but it has to keep track of time. On the other hand, clocks are not mentioned in Definition 3. This omission is intentional. As opposed to a TBA, a real-time algorithm has access to storage space; hence it can use (part of) this storage for time-keeping purposes. The absence of clocks implies that, by contrast to TBA, there are no time constraints on state transitions in a real-time algorithm. Such constraints are, however, made immaterial by the semantics of the input tape. Indeed, an input symbol is not available to the algorithm at a time smaller than the timestamp of that symbol. We believe that this constraint is sufficient as, conforming to Definition 3, time restrictions are imposed by the input itself. Note that real world real-time applications have the same property: their behaviour should conform to time restrictions imposed on their input and/or output rather than internal temporal constraints.

3.3 Comments

The term *real-time* is used complexity theorists in a somewhat different manner than by those in the real-time systems community. Indeed, systems researchers use the term to refer to those computations in which the notion of correctness is linked to the notion of time [5]. By contrast, theorists often use *real-time* as synonym for online or linear time (the real-time Turing machine [1, 2] is probably the best example for the latter use). The model of well-behaved timed ω-languages intends to bridge this gap: although it is a formal model, it captures all the features of real-time computations as understood by the systems community. Therefore, this model has little

in common with the theorists' real-time concept. However, we believe not only that well-behaved timed ω-languages accurately model the notion of real-time computation used in the systems community, but also that our model can be the basis of a meaningful complexity theory of real-time systems.

Claim 1. Well-behaved timed ω-languages model exactly all real-time computations.

We intend to investigate a complexity-theoretical approach to real-time computations. In other words, our intended research direction is to define complexity classes for timed ω-languages, that capture an intuitive notion of real-time efficiency, and study the relations between these classes and between them and existing complexity classes. Classical complexity theory is of central concern not only for theorists but also for practitioners. For example, the existence of a lower complexity bound for some problem is an important point: no matter how clever a program is, the bound cannot be overcome. Practitioners in the real-time systems area do not have such a theory to refer to. As a consequence, any question related to resource allocation and even solvability for a real-time problem is unique, in the sense that the answer to such a question should be developed from scratch (by either experiments or individualized proofs). A complexity-theoretic approach to real-time algorithms should offer a common ground to which practitioners can refer in order to get readily available answers to the mentioned questions.

Suppose we prefix real-time complexity classes by "$rt-$." Then, some possible complexity classes are rt–PROC(f) (parallel $f(n)$-processor real-time algorithms). The relations between these classes can be investigated. For example, an interesting question might be: is the hierarchy rt–PROC(f) infinite? In other words, given any number k of processors, is there a well-behaved timed ω-language that can be accepted by a k-processor real-time algorithm but cannot be accepted by a $(k-1)$-processor one? If the answer is negative, then a practitioner looking at implementing some system might give less consideration to a parallel implementation and focus first on some other options, such as obtaining a timely approximate solution instead of an exact solution that is available too late. On the contrary, should the answer be positive, considering a parallel implementation might become a high priority for the practitioner.

Finally, we note some differences and similarities between timed ω-languages (i.e., real-time algorithms) and classical formal languages (i.e., classical algorithms). On the one hand, it is immediate that formal languages are particular cases of timed ω-languages. Indeed, save for the time sequence, any word is a timed ω-word. If one relies on the semantics of the time sequence, one can add the time sequence $00\cdots0$ to a classical word and obtain the corresponding timed ω-word. However, none of the timed ω-words obtained in this manner is well behaved. We thus have a crisp delimitation between real-time and classical algorithms, while keeping the formalisms as unified as possible.

4. Real-Time Models

Our thesis is that the theory of timed languages covers all the practically meaningful aspects of real-time computations, while doing so in a formal, unified manner. In order to support this thesis, we take some meaningful examples and construct timed ω-languages that model them. First, we model two general concepts that are central to real-time systems, namely, computing with deadlines, and real-time input arrival. The appearance of either of these concepts in the specification of some problem gives the real-time characteristic to that problem [5]. Thus, being able to construct a suitable model for these concepts is crucial to the usefulness of the theory of timed ω-languages. Once models for these concepts are in place, one can use them in order to analyse practical real-time applications. Indeed, we shall further consider two applications and use the results from this section in order to build formal models for them.

Given some problem, we denote the input (output) alphabet by Σ (Ω). We denote by n and m the sizes of the input ι and of the output o. When a timed ω-word is denoted by (σ, τ), we consider that $\sigma = \sigma_1\sigma_2\cdots$ and $\tau = \tau_1\tau_2\cdots$. We consider without loss of generality that Σ, Ω, and $\mathbb{N}$ are disjoint.

4.1 Computing with Deadlines

One of the most often encountered real-time features is the presence of deadlines. The deadlines are typically classified into firm deadlines, when a computation that exceeds the deadline is useless, and soft deadlines, where the usefulness of the computation decreases as time elapses [6].

Let Π be a problem whose instances can be classified into three classes: (1) no deadline is imposed; (2) a firm deadline is imposed at time t_d; (3) a soft deadline is imposed at time t_d and the usefulness function is u: $[t_d, \omega) \to [max, 0]$ after this deadline. We build for each instance a timed ω-word (σ, τ) over $\Sigma \cup \Omega \cup (\mathbb{N} \cap [max, 0]) \cup \{w, d\}$, $w, d \notin \Sigma \cup \Omega$ as follows. (1) $\sigma_1 \cdots \sigma_m = o$, $\sigma_{m+1} \cdots \sigma_{m+n} = \iota$, $\sigma_i = w$ for $i > m+n$, $\tau_i = 0$ for $1 \leq i \leq m+n$, and $\tau_i = i - m - n$ for $i > m+n$. (2) $\sigma_1 \in \mathbb{N} \cap [max, 0)$, $\sigma_2 \cdots \sigma_{m+1} = o$, $\sigma_{m+2} \cdots \sigma_{m+n+1} = \iota$, $\tau_i = 0$ for $1 \leq i \leq m+n+1$; if $\tau_i < t_d$ and $i > m+n+1$, then $\tau_i = i-m-n-1$ and $\sigma_i = w$. Let i_0 be the index such that $\tau_i = t_d$. Then, for all $i \geq i_0$, $\tau_i = i_0 + \lfloor (i - i_0)/2 \rfloor$, and $\sigma_i = d$ if $i - i_0$ is even and 0 otherwise. (3) Same as case (2), except that $\sigma_i = \lfloor u(\tau_i) \rfloor$ whenever $i - i_0$ is odd.

Let the language formed by all the ω-words that conform to the above description be L. Basically, a timed ω-word in L has the following properties: at time 0, a possible output and a possible input for Π are available. Then, up to the deadline d, the symbols that arrive are w. After that, each time unit brings to the input a pair of symbols, the first component being d (signaling that the deadline passed) and the second one being the measure of usefulness the computation still has (which is 0 forever when the deadline is firm). When a deadline is imposed over the computation (cases (2) and (3)), a minimum acceptable usefulness estimate is also present at the

beginning of the computation. Let $L(\Pi)$ be the language of successful instances of Π, $L(\Pi) \subseteq L$, in the sense that an ω-word x from L is in $L(\Pi)$ iff some algorithm that solves Π, when processing the input from x, outputs the output from x either within the imposed deadline (if any), or at a time when the usefulness of the process is not below the acceptable limit from x.

We are now ready to present an acceptor for $L(\Pi)$. For simplicity, we consider that this acceptor is composed of two "processes," P_w and P_m. P_w is an algorithm that solves Π, which works on the input of Π contained in the current input ω-word, and stores the solution in some designated memory space upon termination. If there is more than one solution for the current instance, then P_w nondeterministically chooses that solution that matches the proposed solution contained in the ω-word, if such a solution exists. Meantime, P_m monitors the input. If at the moment P_w terminates the current symbol is w then P_m compares the solution computed by P_w with the proposed solution and imposes on the whole acceptor some "final" state s_f (which is the only state that writes f to the output tape) if they are identical, or some other designated state s_r (for "reject") otherwise. On the other hand, if at the moment P_w terminates the current symbol is d, then the deadline passed. Then P_m compares the current usefulness measure with the minimum acceptable one. If the usefulness is not acceptable, P_m imposes the state s_r on the whole acceptor. Otherwise, P_m compares the result computed by P_w with the proposed solution and imposes either the state s_f or s_r, accordingly. Once in one of the states s_f or s_r, the acceptor keeps cycling in the same state. In state s_f, the acceptor writes f on the output tape. The output tape is not modified in any other state. It is clear that the language accepted by the above acceptor is exactly $L(\Pi)$. It is also clear that $L(\Pi)$ is well behaved. Thus we complete the modelling of computations with deadlines in terms of ω-languages.

4.2 Real-Time Input Arrival

We assumed in the previous section that all the input data are available at the beginning of computation. However, the case when data arrive while the computation is in progress is easily modelled by modifying the timestamps that correspond with each item of input data. One of the computational paradigms that feature real-time input arrival is the data-accumulating paradigm, which has been extensively studied in [7, 8]. The shape of input in this paradigm is very flexible, and any practically important form of real-time input arrival can be modelled by a particular class of problems within this paradigm. Therefore, in what follows we use the data-accumulating paradigm to illustrate the concept of real-time input arrival. Further models of this concept are also presented in Section 5.

A data-accumulating algorithm (d-algorithm) works on an input considered as a virtually endless stream. The computation terminates when all the currently arrived data have been processed before another datum arrives. In addition, the arrival rate of the input data is given by some function $f(n,t)$ (the data arrival law), where n denotes the amount of data that is available beforehand, and t denotes the time. The family of arrival laws most commonly used as examples is $f(n,t) = n + kn^{\gamma}t^{\beta}$, where k, γ, and β are positive constants. Any successful computation of a d-algorithm terminates in finite time.

Given a problem Π pertaining to this paradigm, we can build the corresponding timed ω-language $L(\Pi)$ similarly to Section 4.1: Given some (infinite) input word ι for Π (together with some $f(n,t)$ and some n), and a possible output o of an algorithm solving Π with input ι, a timed ω-word (σ,τ) that may pertain to $L(\Pi)$ is constructed as follows: $\sigma_1 \cdots \sigma_m = o$, $\sigma_{m+1} \cdots \sigma_{m+n} = \iota_1 \cdots \iota_n$, $\tau_i = 0$ for $1 \leq i \leq m+n$. Note that, as both the arrival law and the initial amount of data are known, one can establish the time of arrival for each input symbol ι_j, $j > n$. Let us denote this arrival time by t_j. Also, let $i_0 = m+n+1$. Then the continuation of the timed ω-word is as follows: for all $i \geq 0$, $\sigma_{i_0+2i} = c$ (where c is a special symbol), and $\sigma_{i_0+2i+1} = \iota_{i_0+i}$; moreover, $\tau_{i_0+2i+1} = t_{i_0+i}$ and $\tau_{i_0+2i} = \tau_{i_0+2i+1} - 1$.

Now, an acceptor for $L(\Pi)$ has a structure that is identical[2] to the one used in Section 4.1: it consists in the two processes P_w and P_m. P_w works exactly like the P_w from Section 4.1, except that it emits some special signal to P_m each time it finishes the processing of one item of input data. Note that, as any d-algorithm is an online algorithm [8], it follows that once such a signal is emitted the p-th time, P_w has a (partial) solution immediately available for the input word $\iota_1 \cdots \iota_p$.

Then suppose that P_m received p signals from P_w, and it also received the input symbol $\sigma_{i_0+2(p-1-i_0)}$, but it did not yet receive the input symbol $\sigma_{i_0+2(p-i_0)}$. This is the only case when P_m attempts to interfere with the computation of P_w. In this case, P_m compares the current solution computed by P_w with the solution proposed in the input ω-word; if they are identical, the input is accepted, and otherwise the input is rejected (in the sense that either state s_f or s_r is imposed upon the acceptor, accordingly). It is clear that $L(\Pi)$ is well behaved and contains exactly all the successful instances of Π; therefore we have succeeded in modelling d-algorithms using timed ω-languages.

5. Applications

In Sections 4.1 and 4.2 we modelled the two main ingredients that, when present, impose the real-time qualifier on the problem. This supports our thesis that the theory of timed languages covers all the practically relevant aspects of real-time computations. In order also to emphasize the expressiveness of the formalism, we provide, in what follows, timed ω-languages that model problems from two highly practical areas: real-time databases (where we identify a real-time variant of the recognition problem) and

[2] In particular, if there is more than one solution for the current instance, then P_w nondeterministically chooses that solution that matches the proposed solution contained in the ω-word, if such a solution exists.

ad hoc networks (where we offer a formal model for the routing problem).

5.1 Real-Time Database Systems

We start by briefly reviewing the main concepts of the relational and real-time database systems theory in order to summarize the notations and concepts that are used later, directing the interested reader to [9, 10] for a more detailed presentation. Then we use our formalism for modelling the recognition problem for real-time database systems (RTDBS for short).

Relational databases. Fix two countably infinite sets **att** (of attributes) and **dom** (the underlying domain, disjoint from **att**). A relation is given by its name and its ordered set of attributes (its sort). Given a relation R, the arity of R is $arity(R) = |sort(R)|$. A relation schema is a relation name R. A database schema is a nonempty finite set $\mathbf{R}$ of relation names. Let R be a relation of arity n. A tuple over R is an expression $R(a_1, a_2, \ldots, a_n)$, where $a_i \in \mathbf{att}$, $1 \leq i \leq n$. A relation instance over R is a finite set of tuples over R. A (database) instance $\mathbf{I}$ over some database schema $\mathbf{R}$ is the union of relation instances over $\mathbf{R}$. The sets of instances over a database schema $\mathbf{R}$ (relation schema R) are denoted by $inst(\mathbf{R})$ ($inst(R)$). The interrogation of a database is accomplished by queries. A query q is a partial mapping from $inst(\mathbf{R})$ to $inst(S)$, for fixed database schema $\mathbf{R}$ and relation schema S.

The (data) complexity of queries is defined based on the recognition problem associated with the query [9]. For a query q, the recognition problem is the language $\{enc(\mathbf{I})\$enc(u) | u \in q(\mathbf{I})\}$, where enc denotes a suitable encoding over queries and tuples, and \$ is a special symbol. The (data) complexity of q is the (conventional) complexity of its recognition problem. For each conventional (time, space, processors) complexity class C, one can define a corresponding complexity class of queries QC.

Real-time databases. An active database supports the automatic triggering of updates (rules) in response to (internal or external) events. The typical form of a rule is "on *event* if *condition* then *action*." An action may in turn generate other events and hence trigger other rules. A fundamental issue in active databases addresses the choice of an execution model (i.e., a semantics for rule application), with an important dimension of variation given by the moment the rules are fired: immediate firing (a rule is fired as soon as its event and condition become true), deferred firing (rule invocation is delayed until the final state in the absence of any rule is reached), and concurrent firing (a separate process is spawned for the rule action and is executed concurrently). In the most general model, each rule has an associated firing mode.

RTDBSs add temporal and timeliness dimensions to active databases. Indeed, a real-time database interacts with the physical world, and the database is thus active. In addition, data in a real-time database are time sensitive, and the transactions must be timely, that is, they must complete within their time constraints (deadlines). We briefly present in the following the data model used in [10].

The objects from the database are grouped into three categories. *Image objects* are those objects that contain information obtained directly from the external environment. Associated with an image object is the most recent sampling time. A *derived object* is computed from a set of image objects and possibly other objects. The timestamp associated with a derived object is the oldest valid time of the data objects used to derive it. Finally, an *invariant object* is a value that is constant with time. Then, a real-time database instance is defined as $B = (I_1, I_2, \ldots, I_n, D, V)$, where I_n is the most recent set of image objects and $I_1, I_2, \ldots, I_{n-1}$ are archival variants of this set. D is the set of derived objects, and V is the set of invariant ones. It is assumed that the difference between the valid time and the transaction time is small. Time is considered discrete and linear. The valid time associated with each temporal object in the database instance is called the lifespan of the object. The lifespan of a data object is defined as a finite union of intervals that form a Boolean algebra. A lifespan can also be associated with a set of objects, in a natural manner. Based on these notions, a variant of relational algebra can be defined as a query language for real-time databases [10].

Additional issues in RTDBSs include the pattern of queries (periodic, sporadic, aperiodic), the nature of deadlines (hard, firm, soft), and the way the updating rules are fired. Although the first two issues have received both theoretical and practical attention in the literature, to our knowledge there is no special theoretical treatment of the last issue, except for the one that is a spawn of active database theory. Immediate firing in the case of image objects is implied in [10], and therefore in the above paragraphs.

Some aperiodic query q is a partial function from B to $inst(S)$, where S is some relation. A periodic query returns an answer each time it is issued; therefore such a query is a function from B to $(inst(S))^\omega$.

5.1.1 RTDBSs as Timed Languages

It seems natural to try to model RTDBSs using timed languages. We describe such a modelling in what follows. We consider that there is a suitable encoding function *enc* that encodes objects and sets of objects, without giving much attention to how such a function is constructed, as such functions were widely used. Let \$ be a symbol that is not in the codomain of *enc*.

We ignore queries for the moment. Recall that a real-time database instance is a tuple $B = (I_1, I_2, \ldots, I_n, D, V)$. Assume for now that the database contains exactly one image object o_k, and that the value of o_k is read from the external world each t_k time units. Let D (V) be some set of derived (invariant) objects, with $m = |enc(V)|$ and $p = |enc(D)|$, and $o_k(t)$ be the value of o_k that is read at time t from the external world. Consider then the timed ω-word $db_k = (\sigma, \tau)$, having the following form: let[3] $q = |enc(o_k)|$; then for any

[3] We assume for clarity of presentation that the length of the encoding of o_k is constant over time. The extension to a variable length is straightforward.

$i \geq 0$, $\sigma_{\alpha+i(q+1)+1} \cdots \sigma_{\alpha+(i+1)(q+1)} = enc(o_k(t_i))\$$, where $\alpha = m + p + 2$, and $\tau_j = it_i$ for $\alpha + i(q+1) + 1 \leq j \leq \alpha + (i+1)(q+1)$. Let $db_0 = (\sigma', \tau')$, such that $\sigma'_1 \cdots \sigma'_m = enc(V)$, $\sigma'(m+1) = \sigma'(m+p+2) = \$$, and $\sigma'_{m+2} \cdots \sigma'_{m+p+1} = enc(D)$; in addition, $\tau'_i = 0$, $1 \leq i \leq m + p + 2$.

In other words, the sets of both invariant and derived objects are specified at time 0, as modelled by the word db_0. Then each t_k time units a new value for o_k is provided. This is modelled by db_k. It is clear that the database instance is completely specified by the word $db_0 db_k$, as this word models the invariant and derived objects (by db_0), as well as all the updates for the sole image object (by db_k).

We consider now the general case of a real-time database. That is, let the database instance contain r image objects o_k, $1 \leq k \leq r$. If we consider a word db_k corresponding to each object o_k, $1 \leq k \leq r$, then it is clear that the database is described by the word $db_B = db_0 db_1 \cdots db_r$.

We now have a model for real-time databases, on which is trivially a well-behaved timed ω-language. Now all that we have to do is to consider the queries. Again, we assume without further details that there is a function enc_q for encoding queries and their answers, whose codomain is disjunct from the codomain of enc.

Let us focus on aperiodic queries first. Each such query q may have a firm or soft deadline. However, it seems natural to also consider queries without any deadline, as they might be present even in a real-time environment. Therefore, the encoding of a query should include (a) the time t at which the query is issued, (b) the (encoding of) the query itself $enc_q(q)$, (c) a tuple s that might be included in the answer to the query, and (d) the deadline t_d of the query, if any.

Note that a similar problem is the presence of deadlines, considered in Section 4.1, except that the first item is not modelled (the computation always starts at time 0). Therefore, our construction is similar to the construction of the language that models computations with deadlines. We have thus a query for which (a) there is no deadline, (b) a firm deadline is present, or (c) a soft deadline is present. The deadline (if any) is imposed at some relative time t_d (i.e., the moment in time at which the deadline occur is $t + t_d$), and the usefulness function is denoted by u (u: $[t_d, \infty) \to \mathbb{N} \cap [max, 0]$). For each query q and each candidate tuple s we build an ω-word $aq_{[q,s,t]} = (\sigma, \tau)$ as follows, where $m = |enc_q(s)\$|$, $n = |enc_q(q)\$|$, and $\$, w_q, d_q$ are not contained in the codomain of enc_q: (1) $\sigma_1 \cdots \sigma_m = enc_q(s)\$$, $\sigma_{m+1} \cdots \sigma_{m+n} = enc_q(q)\$$, $\sigma_i = w_q$ for $i > m+n$, $\tau_i = t$ for $1 \leq i \leq m+n$, and $\tau_i = t+i-m-n$ for $i > m+n$. (2) $\sigma_1 \in \mathbb{N} \cap [max, 0)$, $\sigma_2 \cdots \sigma_{m+1} = enc_q(s)\$$, $\sigma_{m+2} \cdots \sigma_{m+n+1} = enc_q(q)\$$, $\tau_i = t$ for $1 \leq i \leq m+n+1$; if $\tau_i < t_d$ and $i > m+n+1$, then $\tau_i = t+i-m-n-1$ and $\sigma_i = w_q$. Let i_0 be the index such that $\tau_i = t + t_d$. Then, for all $i \geq i_0$, $\tau_i = t + i_0 + \lfloor (i - i_0)/2 \rfloor$, and $\sigma_i = d_q$ if $i - i_0$ is even and $\sigma_i = 0$ otherwise. (3) This case is the same as case (2), except that for $i \geq i_0$ $\sigma_i = \lfloor u(\tau_i) \rfloor$ if $i - i_0$ is odd.

Now, let q be a periodic query, issued for the first time at time t and reissued each t_p time units. Each time q is issued, we have to consider a tuple whose inclusion into the result of q is to be tested. Let s_i be such a tuple for the ith invocation of q, and let $s = (s_1, s_2, s_3, \ldots)$. Such a query is modelled by the word $pq_{[q,s,t,t_p]} = aq_{[q,s_1,t]} aq_{[q,s_2,t+t_p]} aq_{[q,s_3,t+2t_p]} \cdots$. It is clear that, for a word $pq_{[q,s,t,t_p]} = (\sigma, \tau)$ and for any finite positive integer k, there exists a finite integer k' such that $\tau_{k'} \geq k$. Thus, $pq_{[q,s,t,t_p]}$ is well behaved.

We have therefore modelled the main ingredients of a RTDBS. All we have to do is to put the pieces together.

Definition 4. Let B be some real-time database instance. Given some aperiodic query q from B to $inst(S)$ (where S is some relation schema), issued at time t, the recognition problem for q on B is the (well-behaved) timed ω-language $L_{aq} = \{db_B aq_{[q,s,t]} | s \in q(B)\}$. Analogously, given a periodic query q from B to $(inst(S))^\omega$, issued at time t and with period t_p, the recognition problem for q on B is the (well-behaved) timed ω-language $L_{pq} = \{db_B pq_{[q,s,t,t_p]} | s \in q(B)\}$.

5.2 *Ad Hoc* Networks

We now direct our attention to another real-time problem, the routing problem in *ad hoc* networks. We show how to model this problem using the theory of timed ω-languages. In the process, we also identify an interesting variant of real-time algorithms, which we believe to be useful in modelling parallel distributed real-time systems.

An *ad hoc* network is a collection of wireless mobile nodes that dynamically forms a temporary network without using any existing infrastructure or centralized administration [11]. In such a network, the set of those nodes that can be directly reached by some host changes with time. Because of this volatility of the set of directly reachable nodes, each node should act not only as a host, but as a router as well, forwarding packets to other hosts in the network.

Although the concept of *ad hoc* networks is relatively new, many routing algorithms have been developed (see, e.g., [11] and the references therein). A comparative performance evaluation was proposed for the first time in [11], where several routing algorithms are compared based on discrete event simulation. To our knowledge, no analytical model have been proposed to date. On the other hand, an *ad hoc* network is obviously a real-time system, similar to the paradigm of correcting algorithms [12]. Therefore, one can model *ad hoc* networks using timed ω-languages. This is what we attempt in the following.

We assume that a message emitted by some node at some time t is received by another node that is in the transmission range of the sender at time $t+1$. We thus establish a granularity of the time domain. This granularity seems appropriate, as transmitting a message is an elementary operation. We say that $range(n_1, n_2, t) = true$ iff node n_2 is in the transmission range of node n_1 at time t.

5.2.1 The Routing Problem

The main component of a model for *ad hoc* networks is the mobile host (or the node). It is consistent to assume

that each node in a network is uniquely identified (e.g., by its IP address). For convenience, we label such a node by an integer between 1 and n, where n is the number of nodes in the given network. We assume that there exists an encoding function e of the properties of any node i (the label i of the node, the position of i, other properties that will be explained below) over some alphabet Σ, with @, \$ $\notin \Sigma$. Denote by Π the set of all possible properties. Then, we say that x is the encoding of some property π of node i iff $x = enc(i, \pi)$, where enc: $\mathbb{N} \times \Pi \rightarrow \Sigma$, $enc(i, \pi) = \$e(i)\$$ if $\pi = i$ and $enc(i, \pi) = \$e(i)@e(\pi)\$$ otherwise. In other words, we have a standard encoding, except that each property of some node i (except i itself) is prefixed by an encoding of i.

Each node i is characterized by its position, which changes with time. Denote by $p_i(t)$ the (encoding of the) position of node i at time t. In addition, each node has a set of characteristics that are invariant with time (e.g., the transmission range). The structure of this set is, however, immaterial, and we consider that these characteristics are encoded by some string q_i for each node i.

We are now ready to consider a timed ω-word that models some mobile host. A node i is modelled by the word $h_i = (\sigma, \tau)$, where $\sigma = (q_i)(\prod_{t=0}^{\omega} p_i(t))$, and $\tau = \tau_1\tau_2\cdots$, with $\tau_j = 0$ for $1 \leq j \leq |q_i p_i(0)|$, and, for any $k > 1$, $\tau_j = k$, $1 + |q_i| + \prod_{l=0}^{k-1} |p_i(l)| \leq j \leq |q_i| + \prod_{l=0}^{k} |p_i(l)|$. In other words, the first part of h_i contains the invariant set of characteristics, together with the initial position of the object that is modelled. The time values associated with this subword are all 0. Then, the successive positions of the node are specified, labelled with their corresponding time values. It is clear that all the necessary information about node i is contained in the word h_i.

Now that we have a model for the set of nodes, all we have to do is to connect them together. We have, that is, to model message exchanges between nodes. Consider a message u issued at some time t. Such a message should contain the source node s and the destination node d. In addition, a message may contain its type (e.g., message or acknowledgment), the data that are to be transmitted, and so on. All this content (referred to as the body of the message) is, however, immaterial, and we denote it by b_u as a whole. Considering that the encoding function e introduced above encodes messages over Σ as well, let the encoding of a message be $\$e(t)@e(s)@e(d)@e(b_m)\$$, $|\$e(t)@e(s)@e(d)@e(b_u)\$| = k$. The timed ω-word that models u is $m_u = (\sigma, \tau)$, where $\sigma_1 \cdots \sigma_k = \$e(t)@e(s)@e(d)@e(b_u)\$$ and $\tau_j = t$ for $1 \leq j \leq k$. Note that m_u is not a well-behaved timed ω-word. However, for any node i, $h_i m_u$ is well behaved. In addition, for a message to exist, there must be at least one node in the network (the sender). That is, a model of a message would always be concatenated to the model of at least one node. The above construction is thus sufficient for our purposes.

One also needs to consider the model for the receiving event. Assume that some message u (generated at time t_u, by source s) is received by its intended destination d at time t'_u. We model such an event by the timed word $r_u = (\sigma, \tau)$, where $\sigma_1 \cdots \sigma_{k'} = \$e(t)@e(s)@e(d)\$$ and $\tau_j = t'_u$ for $1 \leq j \leq k'$, with $k' = |\$e(t)@e(s)@e(d)\$|$. Again, r_u is not well behaved, but the above argument still holds (no acknowledgment exists in a network with no hosts).

An *ad hoc* network with n nodes and without any message is modelled by the timed ω-word $a_n = h_1 h_2 \cdots h_n$. Then, a network of n nodes and some messages $u_1, u_2, \ldots, u_k$, $k \geq 1$ are modelled by the word $w_{n,k} = h_1 h_2 \cdots h_n m_{u_1} m_{u_2} \cdots m_{u_k}$, and the model that includes the event of receiving u_i, $1 \leq i \leq k$ is $wr_{n,k} = h_1 h_2 \cdots h_n m_{u_1} r_{u_1} m_{u_2} r_{u_2} \cdots m_{u_k} r_{u_k}$. Given some countably infinite series of messages $u_1 u_2 \cdots$, the model of the network in which these messages are transmitted is $wr_{n,\omega} = h_1 h_2 \cdots h_n m_{u_1} r_{u_1} m_{u_2} r_{u_2} \cdots$. Note that $w_{n,\omega}$ is a well-behaved timed ω-word under the reasonable assumption that any node can generate only a bounded number of messages per time unit. In the following we may refer to the encoding m_u of a message u simply by "the message m_u," the exact meaning being given by the context. For a fixed n, denote by N_n the set of all words of the form $w_{n,k}$, $k \in \mathbb{N} \cup \{\omega\}$.

We are now ready to state the routing problem in *ad hoc* networks as a timed ω-language. Consider a network with n nodes, and a message u generated at time t, with body b, that is to be routed from its source s to the destination d. Then, a route of u is a word in the timed ω-language $R_{n,u} \subseteq N_n$ where, for some finite positive integer f, there exists a set of messages $u_1, u_2, \ldots, u_f$, and possibly a set of messages $rt_1, rt_2, \ldots, rt_g$, with g a positive, finite integer, such that any $w \in R_{n,u}$ has the form $w = h_1 h_2 \cdots h_n m_{u_1} r_{u_1} \cdots m_{u_f} r_{u_f} m_{rt_1} r_{rt_1} \cdots m_{rt_f} r_{rt_f}$. For each message u_i, $1 \leq i \leq f$, denote by t_i, t'_i, s_i, d_i, and b_i the generation time, receiving time, source, destination, and body of u_i, respectively. Then, (a) $b_1 = b_2 = \cdots = b_f = b$, $s_1 = s$, $d_f = d$, $t_1 = t$; (b) for any i, $1 \leq i \leq f - 1$, $d_i = s_{i+1}$, $t'_i = t_{i+1}$, and $range(s_i, d_i, t_i) = true$; and (c) t'_f is finite.

In other words, the routing process generates f intermediate, one-hop messages $(u_1, \ldots, u_f)$. The time at which one of these messages arrives at the intended destination of u is finite (otherwise, the routing process is unsuccessful). In addition, there might exist a finite number of additional messages $(rt_1, \ldots, rt_g)$, that are exchanges between nodes in the routing process (e.g., when the routing tables at each node are built/updated). In the following, we refer to some language $R_{n,u}$ as an (instance of a) routing problem, and some particular word $w \in R_{n,u}$ will be called an instance of $R_{n,u}$, or just routing instance when $R_{n,u}$ is understood from the context. Note that the actual routing (performed by some routing algorithm) of message u in some n-node network is modelled by a word in the corresponding routing problem.

Clearly, the language $R_{n,u}$ models all the relevant characteristics of a routing problem. Two routing algorithms may be compared by comparing their corresponding words from $R_{n,u}$, and more than one measure of performance may be considered. The measures given in [11] are the routing overhead (the total number of messages transmitted), path optimality (the difference between the number of hops a message took to reach its destination versus the length of

the shortest possible path), and the message delivery ratio (the number of messages generated versus the number of packets received). These measures have a clear correspondent in our model. Indeed, for some $w \in R_{n,u}$, the routing overhead is given by $f + g$, the total number of generated messages; the number of hops is $t'_f - t_1$; and the delivery ratio is $|\rho| - |\{w \in \rho | t'_f - t_1 \leq T\}|$, where $\rho = \bigcup_u R_{n,u}$, and T is some finite threshold.[4]

5.2.2 On Routing Algorithms

To now, we have modelled the routing problem. Such an approach offers a basis for comparing routing algorithms, once the results of these algorithms are modelled as words from $R_{n,u}$. On the other hand, nothing is said about the routing algorithm itself. The immediate variant for such a model takes the form of a real-time algorithm that accepts the language $R_{n,u}$. However, further restrictions to such an acceptor must be imposed: the real-world router consists in n independent algorithms that have limited means of communication. That is, two such nodes can communicate only by messages exchanged between them. Put in another way, a node is unaware of the properties of another node unless it receives a message from (or about) that node. Based on this intuition, we can propose a model for an n-node *ad hoc* network. For specificity, we model a routing instance $w = h_1 h_2 \cdots h_n m_{u_1} r_{u_1} \cdots m_{u_f} r_{u_f} m_{rt_1} r_{rt_1} \cdots m_{rt_f} r_{rt_f}$.

Such a model has n component timed ω-words H_i, $1 \leq i \leq n$, one for each node. Each H_i consists in a "local" component $\mathcal{L}_i$ and a "remote" component $\mathcal{R}_i$, with $\mathcal{L}_i = h_i m_{u_{j_1}} m_{u_{j_2}} \cdots m_{u_{j_x}} m_{rt_{k_1}} m_{rt_{k_2}} \cdots m_{rt_{k_y}}$, where $0 \leq x \leq f$, $0 \leq y \leq g$, $1 \leq j_l \leq f$ for any l, $1 \leq l \leq x$, and $1 \leq k_l \leq g$ for any l, $1 \leq l \leq y$. Moreover, the source of each message u_{j_l} or rt_{k_l} is i. That is, the local component consists only in those messages that are sent by the corresponding node, together with the space coordinates of that node.

Given $\mathcal{L}_i$, for each $j \neq i$, $1 \leq j \leq n$, denote by $M_{i,j}$ the set $\{r_{u_{j_l}} | 1 \leq l \leq x, d_{u_{j_l}} = j\} \cup \{r_{rt_{j_l}} | 1 \leq l \leq y, d_{rt_{j_l}} = j\}$. That is, the set $M_{i,j}$ contains the receiving events for all the messages that are sent from node i to node j. Then $\mathcal{R}_i = v_1 \cdots v_k$, where $v_1, \ldots, v_k$ are exactly all the elements in the set $\cup_{l=1}^{n} M_{l,i}$.

Finally, $H_i = \mathcal{L}_i \mathcal{R}_i$. In other words, the component H_i contains only those messages that are sent by the corresponding node and those messages that are received by the node. Besides this information, no knowledge about the external world exists.

6. Distributed Models

We presented, in Section 5.2.2, a distributed model for the routing algorithm. However, such a model can be extended for any distributed real-time algorithm.

The parallelism is introduced in the theory of timed languages by the structure of their acceptors. Indeed, an implementation for such an acceptor can be considered for any underlying model of computation. However, if we use this approach, no information about the explicit distributed character of the computation is present in the language itself. As our goal is to present a consistent theory of real-time computations, it is desirable to have such information right within the language.

A similar construction was studied in the context of conventional languages, namely, the parallel communicating grammar systems (PCGS) introduced in [13] and further studied in, for example, [14, 15]. A PCGS consists in a number of grammars, with their own workspace, that communicate to each other by means of special symbols. Except for this communication, the grammars work independently. The case of parallel grammar systems closely resembles a real-world *ad hoc* network.[5] Based on this intuition, we can propose a model for parallel real-time computations:

Consider a parallel algorithm with n processing elements. In general, one can assume that the implementation is composed of a set of n processes that execute independently and communicate with each other by messages. Consider now some process k isolated from the external world. It has to perform some real-time task; therefore, its execution can be modelled by some timed ω-word. Call this word c_k. In addition to this computation, the process may send messages to other processes. Let all these messages be modelled by some timed ω-word l_k. Furthermore, the messages that are received by process k can be modelled by a timed ω-word r_k. Then, the behaviour of process k is modelled by the timed ω-word $c_k l_k r_k$, $1 \leq k \leq n$.

If some real-time algorithm consists in p such processes, then its behaviour is modelled by the tuple $(c_1 l_1 r_1, \ldots, c_p l_p r_p)$. We believe that once the theory for timed ω-languages is in place, the study of the distributed model presented above is worthwhile.

7. Towards a Complexity Theory

We believe that the notions of timed languages and real-time algorithms as introduced in Section 3 are important tools in developing a complexity theory for real-time systems, which simply does not exit at this time. In this work we presented a general definition of this class of languages and suggested that this definition is powerful enough to model all the practically important aspects of real-time computations. We supported our thesis with meaningful examples.

Besides validating the thesis, the examples offered some interesting insights into the theory of real-time systems. Specifically, we constructed a recognition problem for queries in a RTDBS. Although query complexity issues in traditional database systems have been studied [9], the real-time domain has to our knowledge received no attention. The analysis of complexity of queries in this

[4] In practice, an infinite delivery time usually means that the delivery time exceeds some finite threshold T.

[5] It should be noted, however, that although grammar systems are generative devices, the discussion here instead focuses on accepting devices. Therefore, PCGSs will be taken exclusively as an intuitive support.

domain could be based on the newly developed recognition problem, which is yet another argument in favour of the mentioned complexity theory.

We also presented a model for the routing problem in *ad hoc* networks. Not only did we formalize this problem, opening the road for a complexity analysis of it, but we also identified a variant of our model, suitable for modelling distributed real-time computations. As there is growing practical interest in distributed computations, such a model could be of interest. In particular, it offers an alternative to the real-time producer/consumer paradigm [3], one that is not restricted to periodic message generation. Incidentally, note that current developments in the area of wireless communications are tremendous, and this stresses the importance of theoretical analysis of routing algorithms in *ad hoc* networks, as such an analysis is not affected by fast-changing technological characteristics.

We believe that this article offers the basis (as well as the the motivation) for the development of a complexity theory for real-time systems. In general, such a theory takes into account the measurable resources used by an algorithm, the most important of these being time and space. In the real-time environment, however, time complexity makes little sense, because in most applications the time properties are established beforehand. But as supercomputing is now a reality, a complexity hierarchy for real-time computations with respect to the number of available processors is a very interesting direction, with promising prospects. (Recall here that it has already been established that a parallel approach can make the difference between success and failure [7, 8, 12, 16] or can enhance significantly the quality of solutions [17, 18] in real-time environments.)

We note in closing that similar research was pursued in [2], where the hierarchy was established with respect to the number of tapes of real-time Turing machines. However, on the one hand, a multitape Turing machine is probably not equivalent to a multiprocessor device, and on the other hand, as the real-time domain is a highly practical issue, we think that the use of models closer to real machines (e.g., the PRAM) is desirable. The theory of well-behaved timed ω-languages offers a foundation for this pursuit.

Acknowledgment

This research was supported by the Natural Sciences and Engineering Research Council of Canada.

References

[1] H. Yamada, Real-time computation and recursive functions not real-time computable, *IRE Transactions on Electronic Computers*, *EC-11*, 1962, 753–760.

[2] A.L. Rosenberg, Real-time definable languages, *Journal of the ACM*, *14*, 1967, 645–662.

[3] K. Jeffay, The real-time producer/consumer paradigm: A paradigm for the construction of efficient, predictable real-time systems, *Proc. of the 1993 ACM/SIGAPP Symp. on Applied Computing: States of the Art and Practice*, 1993, 796–804.

[4] R. Alur & D.L. Dill, A theory of timed automata, *Theoretical Computer Science*, *126*, 1994, 183–235.

[5] Comp.realtime: Frequently asked questions (Version 3.4, May 1998), http://www.faqs.org/faqs/realtime-computing/faq/.

[6] M.R. Lehr, Y.-K. Kim, & S.H. Son, Managing contention and timing constraints in a real–time database system, *Proc. of the 16th IEEE Real-Time Systems Symp.*, Pisa, Italy, 1995, 332–341.

[7] F. Luccio & L. Pagli, Computing with time-varying data: Sequential complexity and parallel speed-up, *Theory of Computing Systems*, *31*, 1998, 5–26.

[8] S.D. Bruda & S.G. Akl, The characterization of data-accumulating algorithms, *Theory of Computing Systems*, *33*, 2000, 85–96. For a preliminary version see http://www.cs.queensu.ca/home/bruda/www/data_accum2.

[9] S. Abiteboul, R. Hull, & V. Vianu, *Foundations of databases* (Reading, MA: Addison-Wesley, 1995).

[10] S.V. Vrbsky, A data model for approximate query processing of real-time databases, *Data and Knowledge Engineering*, *21*, 1997, 79–102.

[11] J. Broch, D.A. Maltz, D.B. Johnson, Y.-C. Hu, & J. Jetcheva, A performance comparison of multi-hop wireless *ad hoc* network routing protocols, *4th Annual ACM/IEEE International Conf. on Mobile Computing and Networking*, Dallas, TX, 1998, 85–97.

[12] S.D. Bruda & S.G. Akl, A case study in real-time parallel computation: Correcting algorithms, *Journal of Parallel and Distributed Computing*, *61*, 2001, 688–708. For a preliminary version see http://www.cs.queensu.ca/home/bruda/www/c-algorithms.

[13] G. Păun & L.Sântean, PCGS: The regular case, *Ann. Univ. Buc., Matem. Inform. Series*, *38*, 1989, 55–63.

[14] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, & G. Păun, *Grammar systems: A grammatical approach to distribution and cooperation* (London: Gordon & Breach, 1994).

[15] S.D. Bruda, On the computational complexity of context-free parallel communicating grammar systems, in G. Păun & A. Salomaa (Eds.), *New trends in formal languages*, Springer Lecture Notes in Computer Science 1218 (Springer-Verlag, 1997).

[16] S.G. Akl & L. Fava Lindon, Paradigms admitting superunitary behaviour in parallel computation, *Parallel Algorithms and Applications*, *11*, 1997, 129–153.

[17] S.G. Akl & S.D. Bruda, Parallel real-time optimization: Beyond speedup, *Parallel Processing Letters*, *9*, 1999, 499–509. For a preliminary version see http:// www.cs.queensu.ca/home/akl/techreports/beyond.ps.

[18] S.G. Akl, Nonlinearity, maximization, and parallel real-time computation, *Proc. of the 12th Conf. on Parallel and Distributed Computing and Systems*, Las Vegas, NV, 2000.

Biographies

Stefan D. Bruda is currently a Ph.D. student in the Department of Computing and Information Science at the Queen's University, Kingston, Ontario, Canada. His research interests include computational complexity, parallel and real-time computations, and formal languages. He has co-authored over 10 journal and conference papers in these areas.

Selim G. Akl is Professor of Computing and Information Science at the Queen's University, Kingston, Ontario, Canada. His research interests are in parallel computation. He is the author of *Parallel sorting algorithms* (Academic Press, 1985), *The design and analysis of parallel algorithms* (Prentice-Hall, 1989), and *Parallel computation: Models and methods* (Prentice-Hall, 1997); and the co-author of *Parallel computational geometry* (Prentice-Hall, 1992). He is an editor of *Computational geometry* (Elsevier), *Parallel processing letters* (World Scientific Publishing), and *Parallel algorithms and applications* (Gordon and Breach).