Distributed real-time automata

Cătălin Dima
Bucharest University, Department of Fundamentals of Computer Science,
str. Academiei 14, Bucharest, Romania, R0-70109,
e-mail: cdima@funinf.math.unibuc.ro

Abstract

We introduce a class of automata with real-time constraints, automata which can be seen as tuples of real-time automata working on their own tape and synchronized on their transitions. Though the general class is equivalent to timed automata and hence is not closed under language complementation, the subclass consisting of automata with a so-called *stuttering-free* condition is shown to be closed under language complementation.

1 Introduction

Automata theory remains one of the most fruitful and influential domains of theoretical computer science. Its use ranges from linguistics to semantics and formal verification and its influences can be traced in domains as far as logic and control theory.

In the past decade much interest has been shown in adapting automata theory for the problems of real-time systems, the accent being put on generalizing finite automata by taking time into account.

A few types of automata asserted themselves as useful, but unfortunately their properties are rather complementary. Yet no class of automata can claim to be the most natural and expressive extension of finite automata. Timed automata of [AD94] have an undecidable universality problem, event-clock automata [AFH94, HRS98] though complementable, have rather complicated algebraic properties and are not closed under substitution and real-time automata [Di99] cannot support giving semantics of real-time concurrent programming languages.

In this paper we build upon an idea that arose in [Di00], that a tuple of real-time automata working on their own tape but synchronized on their transitions might still be closed under complementation. This idea has its roots in the semantics of real-time synchronous programming languages which (when working on variables with finite domains) can be given in the so-called Simple Duration Calculus [Dia].

From the very beginning it is apparent that the automata built this way are too powerful - they are equivalent to timed automata of [AD94] and hence the complementation procedure from [Di00] is lost. However we find that by the addition of the so-called *stuttering freeness* condition automata become determinizable and hence complementable. It is the same condition that on real-time automata required some properties of the Kleene algebra of sets of intervals; see [Di00].

The paper runs as follows: in the next section we remind the definitions and properties of signals and real-time automata. The third section contains the definitions concerning distributed real-time automata and the observation that stuttering implies nonclosure under complementation. The fourth section contains the constructions for determinization and complementation of the automata in discussion. We end with a formalization of the problem of constructing the semantics of real-time synchronous programs from the semantics of each component.

2 Preliminaries

Fix some notations first: $\mathbb{R}_+$ denotes the set of nonnegative numbers while $Intv_{\mathbb{Q}_+}$ the set of intervals having nonnegative rational bounds. We denote π_i the usual projection of a cartesian product on the i-th component $\pi_i : \prod_{j=1}^{n} A_j \longrightarrow A_i$, $\pi_i(a_1, \ldots, a_n) = a_i$ and $Diag(A)$ the diagonal relation on A, i.e. $Diag(A) = \{(a,a) \mid a \in A\}$.

For a function $f : \mathbb{R}_+ \longrightarrow A$ and for some $\alpha > 0$, if there exist some $a, b \in A$, $a \neq b$ and $\epsilon > 0$ such that $f(x) = a$ for all $x \in (\alpha - \epsilon, \alpha)$ and $f(x) = b$ for all $x \in (\alpha, \alpha + \epsilon)$ then we call a the *left limit* of f at α and b the *right limit* of f at α and denote them $f(\alpha - 0)$, respectively $f(\alpha + 0)$.

We say that f *has a discontinuity at* α iff $f(\alpha - 0) \neq f(\alpha + 0)$. The discontinuity is *left* iff we also have that $f(\alpha) = f(\alpha + 0)$.

Definition 2.1 *A* ***signal*** *over an alphabet* Σ *is a function* $\sigma : [0, \alpha) \longrightarrow \Sigma$ *whose domain is an initial interval of the positive numbers and which has finitely many discontinuities, all being left discontinuities.*

Given n*-sets of symbols* $\Sigma_1, \ldots, \Sigma_n$ *(not necessarily pairwise disjoint) an* n-***signal*** *is a signal over* $\Sigma_1 \times \ldots \times \Sigma_n$.

We assume throughout this paper that the alphabets are finite. We denote the domain of σ as $dom(\sigma)$ and the endpoint of this domain as $endp(\sigma)$. The set of all signals over Σ is denoted $Sig(\Sigma)$ and subsets of it are called *real-time languages*, in analogy with sets of words over Σ. Also the set of n-signals is denoted $Sig(\Sigma_1, \ldots, \Sigma_n)$. Note that for each $i \in [n]$, the projection $\pi_i : \Sigma_1 \times \ldots \times \Sigma_n \longrightarrow \Sigma_i$ extends to a function denoted π_i too from $Sig(\Sigma_1, \ldots, \Sigma_n)$ to $Sig(\Sigma)$ and defined as $endp(\pi_i(\sigma)) = endp(\sigma)$ and $\pi_i(\sigma)(t) = \pi_i(\sigma(t))$ for all $t \in dom(\sigma)$.

Let now $\sigma \in Sig(\Sigma)$. For each discontinuity α of σ we define the *duration of the symbol after* α as the difference $d(\sigma, \alpha) = \alpha - \beta$ where β is the next discontinuity after α, if it exists, or $endo(\sigma)$ otherwise. Similarly, define $d(\sigma, \alpha)$ for $\alpha = 0$ too. Then we extend this definition for n-signals: the *duration of the i-symbol after* α in an n-signal σ is $d_i(\sigma, \alpha) = d(\pi_i(\sigma), \alpha)$, if α is a discontinuity of $\pi_i(\sigma)$ and undefined otherwise.

$Sig(\Sigma)$ can be endowed with a noncommutative monoidal structure by defining a concatenation operation: given two signals σ_1 and σ_2 define their concatenation as the signal σ with domain $[0, endp(\sigma_1) + endp(\sigma_2))$ and whose value is:

$$\sigma(t) = \begin{cases} \sigma_1(t) & \text{iff } t \in [0, endp(\sigma_1)) \\ \sigma_2(t - endp(\sigma_1)) & \text{iff } t \in [endp(\sigma_1), endp(\sigma_1) + endp(\sigma_2)) \end{cases}$$

We denote the concatenation of σ_1 with σ_2 as $\sigma_1; \sigma_2$. The unit of concatenation is the *unique* signal with empty domain $\sigma_\epsilon : [0, 0) \longrightarrow \Sigma$.

Then $\mathcal{P}(Sig(\Sigma))$, the powerset of $Sig(\Sigma)$, can be given a structure similar to $\mathcal{P}(\Sigma^*)$, namely a *Kleene algebra structure* [Con71] by defining the star of a real-time language $L \subseteq Sig(\Sigma)$ as $L^* = \bigcup_{n \in \mathbb{N}} L^n$ where $L^0 = \{\sigma_\epsilon\}$ and $L^{n+1} = L^n; L$ ($n \in \mathbb{N}$). Here, concatenation on languages is the natural extension of the concatenation on signals.

Definition 2.2 *A* ***real-time automaton*** *(RTA for short) is a tuple $\mathcal{A} = (Q, \Sigma, \lambda, \iota, \delta, S, F)$ where Q is the (finite) set of* states, *$\delta \subseteq Q \times Q$ is the* transition relation, *$S, F \subseteq Q$ are the sets of* initial, *resp.* final *states, $\lambda : Q \longrightarrow \Sigma$ is the* state labeling function *and $\iota : Q \longrightarrow Intv_{\mathbb{Q}_+}$ is the* interval labeling function.

RTA work over signals: a *run* of length n is a sequence of locations $(q_i)_{i \in [n]}$ connected by δ, i.e. $(q_{i-1}, q_i) \in \delta, \forall i \in [n]$. A run is called *accepting* iff it starts in S and ends in F. The run **accepts** a signal σ iff the signal may be splitted into n parts, such that the state within the i-th part equals

$\lambda(q_i)$ while the length of this i-th part is in the interval $\iota(q_i)$. Formally there exist $e_i \in \iota(q_i)$ such that $dom(\sigma) = [0, \sum_{i=1}^{n} e_i)$ and $\sigma(t) = \lambda(q_i)$ for all $t \in [\sum_{j=1}^{i-1} e_j, \sum_{j=1}^{i} e_j)$ and all $i \in [n]$.

This definition may also be extended to any runs and signals: we say that a run is *associated* to a signal iff the above properties hold for the respective run and signal.

Clearly the splitting points must contain all the time points where state changes occur but they might be more splitting points than discontinuities as two consecutive locations in the run might have the same state label. This is a form of nondeterminism specific to automata with real-time constraints.

Definition 2.3 *A RTA $\mathcal{A}$ is* ***language deterministic*** *iff each signal in $L(\mathcal{A})$ is associated to a unique run. $\mathcal{A}$ is* ***stuttering-free*** *iff it does not have a transition (q, r) with $\lambda(q) = \lambda(r)$. $\mathcal{A}$ is* ***state-deterministic*** *iff initial locations have disjoint labels and transitions starting in the same locations have disjoint labels too, i.e. whenever we have that $r \neq s$ and either $r, s \in S$ or $(q, r), (q, s) \in \delta$ then $\lambda(r) \neq \lambda(s)$ or $\iota(r) \cap \iota(s) = \emptyset$. $\mathcal{A}$ is simply called* ***deterministic*** *whenever it is both state-deterministic and stuttering-free.*

Hence for stuttering-free RTA we have the property that the splitting points are unique for each signal. Notice also that there might be state-deterministic RTA that are not language deterministic due to stuttering steps. The important property that stuttering-free RTA have is that the usual determinization procedure of finite automata can be adapted for them:

Theorem 2.4 *The class of languages which are accepted by stuttering-free real-time automata is closed under complementation.*

This theorem is a corollary of the results proven in [Di00].

3 Distributed real-time automata

Definition 3.1 *An n-**distributed real-time automaton** (n-DRTA) is a tuple*

$$\mathcal{A} = (Q, \Sigma_1, \ldots, \Sigma_n, \delta, \lambda_1, \ldots, \lambda_n, \iota_1, \ldots, \iota_n, S, F)$$

where $n \geq 1$, Q is the set of states, *$\lambda_i : Q \longrightarrow \Sigma_i$ are the* state labeling functions, *$\iota_i : Q \longrightarrow Intv_{\mathbb{Q}_+}$ are the* interval labeling functions, *$\delta \subseteq Q \times \mathcal{P}([n]) \times Q$ is the* transition relation *and $S, F \subseteq Q$ are respectively the sets of* initial, *resp.* final *states. The* transitions *$(q, K, r) \in \delta$ also obey the following condition: if $\lambda_i(q) \neq \lambda_i(r)$ then $i \in K$.*

The second component of each transition is called the *resetting component*. Its use will be apparent in defining the language of these automata.

A *run* is some alternating sequence $(q_1, K_1, q_2, \ldots, K_{m-1}, q_m)$ consisting of states $q_i \in Q$ and sets of indices $K_i \subseteq [n]$ having the property that $(q_j, K_j, q_{j+1}) \in \delta$ for all $j \in [m-1]$. An *accepting run* is then a run that begins in an initial state and ends in a final state. We denote $\text{Runs}(\mathcal{A})$ the set of runs of $\mathcal{A}$ and $\text{ARuns}(\mathcal{A})$ the set of accepting runs of $\mathcal{A}$.

Intuitively, in n-DRTA there are n real-time automata which work in parallel but synchronized on certain transitions. The resetting component is used by each RTA to determine where its current state has to be changed. These ideas are formalized as follows: the *i-th underlying RTA* of the above n-DRTA is the RTA $\mathcal{A}_i = (Q, \Sigma_i, \delta_i, \lambda_i, \iota_i, S, F)$ where

$$\begin{aligned} \delta_i \;\; = \;\; & \{(q,r) \in Q^2 \mid \exists (q = q_1, K_1, q_2, \ldots, K_{m-1}, q_m = r) \in \text{Runs}(\mathcal{A}) \text{ and} \\ & \exists j_0 \in [m-1] \text{ s.t. } i \in K_{j_0} \text{ and } i \notin K_j,\ \forall j \in [m-1], j \neq j_0\} \end{aligned} \tag{1}$$

Then, given a run $\rho = (q_1, K_1, q_2, \ldots, K_{m-1}, q_m)$ we may define the *i-th projection* of this run as the set of runs in $\mathcal{A}_i$ induced by ρ:

$$\begin{aligned} pr_i(\rho) \;\; = \;\; & \{(q_{i_j})_{j \in [k]} \mid k \leq m,\ i_j \in [m],\ i_1 = 1, i_k = m \text{ and} \\ & (q_{i_j}, q_{i_{j+1}}) \in \delta_i\ \forall j \in [k-1]\} \end{aligned}$$

Note that they might be more than one run in $pr_i(\rho)$ as the definition 1 implies that a single transition in δ induces a whole range of transitions in δ_i. We also call the runs in $pr_i(\rho)$ *unidimensional*.

Then, given an n-signal $\sigma \in Sig(\Sigma_1, \ldots, \Sigma_n)$, we say it is *accepted* by the accepting run $\rho = (q_1, K_1, q_2, \ldots, K_{m-1}, q_m)$ (and hence *accepted* by the n-DRTA) iff for each $i \in [n]$ the signal is accepted by some unidimensional run in the i-th projection of ρ. The *language accepted* by $\mathcal{A}$ is the set of n-signals accepted by $\mathcal{A}$. We say two n-DRTA are *equivalent* iff they have the same language. We will denote by **DRTA** the class of all n-DRTA, for all $n \in \mathbb{N}$.

We may observe here that "some" may be replaced with "any" in the definition of acceptance of a signal. Also

Proposition 3.2 *The language emptiness problem is decidable for* **DRTA**.

This follows since n-DRTA are special cases of timed automata with n-clocks which have a decidable emptiness problem [AD94].

Definition 3.3 *A n-DRTA $\mathcal{A}=(Q,\Sigma_1,\ldots,\Sigma_n,\delta,\lambda_1,\ldots,\lambda_n,\iota_1,\ldots,\iota_n,S,F)$ is called* ***language deterministic*** *iff each n-signal $\sigma \in L(\mathcal{A})$ is associated to a unique accepting run.*

$\mathcal{A}$ is called ***state deterministic*** *iff the following conditions are met:*

- *for each pair of transitions $(q,K,r),(q,K,s) \in \delta$ with the property that $\lambda_i(r) = \lambda_i(s)$ for all $i \in [n]$ there is some $j \in [n]$ such that $\iota_j(r) \neq \iota_j(s)$.*
- *for each pair of initial states $r,s \in S$ with the property that $\lambda_i(r) = \lambda_i(s)$ for all $i \in [n]$ there is some $j \in [n]$ such that $\iota_j(r) \neq \iota_j(s)$.*

The automaton is called ***stuttering free*** *iff for each transition $(q,K,r) \in \delta$ K is the set of indices i such that $\lambda_i(q) \neq \lambda_i(r)$.*

If $\mathcal{A}$ is both stuttering-free and state deterministic then it is simply called ***deterministic***.

Finally, the automaton is called ***complete*** *iff for each n-signal (not necessarily in the language of $\mathcal{A}$) there exists a unique associated run that starts in an initial state.*

Hence in a stuttering-free n-DRTA the transition relation could just be given as a *subset of* Q^2 with the property that if $(q,r) \in \delta$ then there exists some $i \in [n]$ such that $\lambda_i(q) \neq \lambda_i(r)$. We will use this definition of the transition relation for these automata. This has also the advantage that runs can be described then simply as (finite) sequences of states $(q_i)_{i\in[m]}$.

It is clear that a deterministic n-DRTA is language deterministic. It is also easy to find some language nondeterministic automata that are still deterministic or some state deterministic automata that are not language deterministic. All these notions are important for proving the closure under complementation: for the usual complementation construction, one has to get a language deterministic automaton which is also complete and then to complement its set of final states.

Note that if the n-DRTA is stuttering-free then all its projections are stuttering-free too.

3.1 Comments on the stuttering-free condition

The stuttering-free condition looks rather strong and as one can easily see it does not allow **DRTA** to be closed under concatenation of languages. However we will find it essential in the determinization procedure: without the stuttering condition n-DRTA have essentially the same expressive power as timed automata of [AD94], hence they have an undecidable universality problem.

It is mentioned in [AD94] that the following language[1] is accepted by some timed automaton, while its complement cannot be:

$$\begin{aligned} L &= \{\sigma \in Sig(\{a,b\}) \mid \exists t \in dom(\sigma) \text{ such that } \sigma(t-0) = a, \sigma(t+0) = b, \\ &\quad \sigma((t+1)-0) = a \text{ and } \sigma((t+1)+0) = b\} \end{aligned}$$

The language consists of signals in which there exist two (possibly non consecutive) discontinuities where the signal jumps from a to b which are separated by a unit interval.

Consider the language $\overline{L}$ consisting of 2-signals over $\{a,b\} \times \{c\}$ whose first projection is L. We can easily find a 2-DRTA for this language:

$$\mathcal{A} = ([6], \{a,b\}, \{c\}, \delta, \lambda_1, \lambda_2, \iota_1, \iota_2, \{1\}, \{6\}) \text{ where}$$

$$\lambda_1(1) = \lambda_1(3) = \lambda_1(5) = a, \quad \lambda_1(2) = \lambda_1(4) = \lambda_1(6) = b,$$
$$\lambda_2(i) = c \text{ for all } i \in [6],$$

$$\iota_1(i) = (0,\infty) \text{ for all } i \in [6],$$
$$\iota_2(1) = \iota_2(2) = \iota_2(5) = \iota_2(6) = (0,\infty), \quad \iota_2(3) = \iota_2(4) = [1,1]$$

$$\begin{aligned} \delta &= \{(1,\{1\},2),(2,\{1\},1),(1,\{1,2\},4),(2,\{1,2\},3),(3,\{1\},4), \\ &\quad (4,\{1\},3),(3,\{1,2\},6),(4,\{1,2\},5),(5,\{1\},6),(6,\{1\},5)\} \end{aligned}$$

Hence 2-DRTA cannot be closed under complementation, otherwise there would have been some timed automata accepting the complement of L.

4 Complementation of stuttering-free n-DRTA

The determinization construction has the aim that each signal has a unique run that accepts it. For stuttering-free n-DRTA this translates to the fact that at each discontinuity there must be only one choice to make between the different transitions enabled at that point.

Let's call a transition (q,r) *enabled* at some discontinuity α in an n-signal σ iff the right limit of each projection of the signal is consistent with the labels of r. It is clear that even for deterministic n-DRTA at each discontinuity of σ there might be more transitions enabled. But the length of the next i-symbol uniquely determines the transition which is to be taken. Therefore in the classical determinization procedure we need to take into

[1] Actually this is a modification of the example in [AD94], which was stated for the timed words semantics of timed automata.

account not only sets of states which have the same state label, but also which share a specific interval label which is not available to any other (similarly state-labeled) set of states. This idea is formalized in the sequel.

Start with a n-DRTA $\mathcal{A} = (Q, \Sigma_1, \dots, \Sigma_n, \delta, \lambda_1, \dots, \lambda_n, \iota_1, \dots, \iota_n, S, F)$ which is stuttering free. Construct the set of subsets of Q consisting of identically labeled states:

$$Id(Q) = \{T \subseteq Q \mid \forall i \in [n]\ \exists a_i \in \Sigma_i \text{ such that } \forall q \in T, \lambda_i(q) = a_i\}$$

Also for each $T \in Id(Q)$ and $T' \subseteq T$ denote $I_i(T, T')$ the intersection of all intervals which label T' together with the complements of the intervals which label $T \setminus T'$:

$$I_i(T, T') = \bigcap_{q \in T'} \iota_i(q) \cap \bigcap_{q \in T \setminus T'} (\mathbb{R} \setminus \iota_i(q))$$

As usual, the intersection of an empty family of intervals is $\mathbb{R}$.

Note that the collection of all sets $I_i(T, T')$, where T is fixed but T' is ranging over all subsets of T forms a *partition* or $\mathbb{R}$. This property is essential in the determinization construction and moreover it assures that the deterministic automaton is *complete*.

On the other hand it's clear that $I_i(T, T')$ may not be an interval, but rather a finite union of intervals. Denote then $int(I_i(T, T'))$ the pair-wise disjoint and *minimal* (in cardinality) set of intervals of which $I_i(T, T')$ is composed. Denote $p_i(T, T')$ the cardinal of $int(I_i(T, T'))$ and consider also some enumeration of the intervals in $int(I_i(T, T'))$: $int(I_i(T, T')) = \{J^i_1, \dots, J^i_{p_i(T,T')}\}$.

The states of the deterministic n-DRTA which is equivalent to $\mathcal{A}$ are $(n+2)$-uples $(T, T', k_1, \dots, k_n)$ where $T \in Id(Q)$, $T' \subseteq T$ and $k_i \in [p_i(T, T')]$ for all $i \in [n]$. Denote this set of tuples $\overline{Q}$. The state labeling functions are then clear,

$$\overline{\lambda}_i : \overline{Q} \longrightarrow \Sigma_i, \qquad \overline{\lambda}_i((T, T', k_1, \dots, k_n)) = \overline{\lambda}_i(q) \text{ for some } q \in T \tag{2}$$

The interval labeling functions are defined using $int(I_i(T, T'))$:

$$\iota_i(T, T', k_1, \dots, k_n) = J^i_{k_i} \quad \forall i \in [n]$$

The set of initial states $\overline{S}$ consists of all the tuples $(S, S', k_1, \dots, k_n)$ in which the set of initial states S is paired with all its *nonempty subsets* S' and some tuple of indices $k_i \in [p_i(S, S')]$. Similarly the set of final states $\overline{F}$ consists of all the tuples $(U, U', k_1, \dots, k_n)$ where U' has a nonempty intersection with the set of final states F (and therefore is itself nonempty) and $k_i \in [p_i(U, U')]$.

For defining the transitions of $\overline{\mathcal{A}}$ we need another notation: for each $(T, T', k_1, \ldots, k_n) \in \overline{Q}$ denote $R(T, T', k_1, \ldots, k_n)$ the element of $Id(Q)$ which collects the destinations of all transitions that start from T' and lead to identically labeled states:

$$\begin{aligned} R(T, T', k_1, \ldots, k_n) &= \{r \in Q \mid \overline{\lambda}_i(r) = a_i \text{ for some fixed } a_i \in \Sigma_i \text{ and} \\ &\quad \exists q \in T' \text{ s.t. } (q, r) \in \delta\} \end{aligned}$$

Then $\overline{\mathcal{A}}$ will have as its transitions all the tuples of the kind

$$((T, T', k_1, \ldots, k_n), (R, R', l_1, \ldots, l_n))$$

where $R = R(T, T', k_1, \ldots, k_n)$, $R' \subseteq R$ and $l_i \in [p_i(T, T')]$. Hence:

Theorem 4.1 *Stuttering-free n-DRTA and deterministic n-DRTA have the same expressive power.*

Yet another important outcome of the above construction is that the n-DRTA obtained is *complete*. The n-DRTA accepting $Sig(\Sigma_1, \ldots, \Sigma_n) \setminus L(\mathcal{A})$ will be then the n-DRTA whose components are the same as for $\overline{\mathcal{A}}$ with the exception of the set of final states which is $\overline{Q} \setminus \overline{F}$. Hence:

Theorem 4.2 *The class of languages which are accepted by n-DRTA is closed under complementation.*

5 Constructing the semantics of real-time synchronous programs

As noted in [Di99], RTA may be used for giving semantics for sequential real-time programs, i.e. where some real-time constraints are put on the delays between the executions of instructions. We can then define the semantics of an n-component synchronous real-time program as a suitable n-DRTA whose projections are the n RTAs that are the semantics of each component but also whose transition relation models the synchronizations between the components. This idea is formalized as follows:

Definition 5.1 *Given n* stuttering-free *RTA $\mathcal{A}_i = (Q_i, \Sigma_i, \lambda_i, \iota_i, \delta_i, S_i, F_i)$, a **synchronization pattern** on $(\mathcal{A}_i)_{i \in [n]}$ is a set of tuples $\delta \subseteq \left(\prod_{i=1}^n Q^i\right)^2 \setminus Diag(\prod_{i=1}^n Q^i)$ with the following property:*

$$((q_1, \ldots, q_n), (r_1, \ldots, r_n)) \in \delta \Rightarrow (q_i, r_i) \in \delta_i \cup Diag(Q_i)$$

A synchronization pattern δ is just a special case of stuttering-free n-DRTA:

$$\mathcal{A} = \left(\prod_{i=1}^{n} Q_i, \Sigma_1, \ldots, \Sigma_n, \lambda_1, \ldots, \lambda_n, \iota_1, \ldots, \iota_n, \prod_{i=1}^{n} S_i, \prod_{i=1}^{n} F_i \right)$$

Therefore the accepting condition and language of a synchronization pattern are particular cases of the respective definitions for n-DRTA.

However the reverse relationship is unclear because it is unclear if each n-DRTA is equivalent to some synchronization pattern of its underlying RTAs. The choice of a synchronization pattern consisting of all $2n$-uples of the form $((q, \ldots, q), (r, \ldots, r))$ is wrong since it might happen that $(q, r) \notin \delta_i$ because it might be the case that $i \notin K$, for any transition (q, K, r) of the given n-DRTA. Hence we conjecture that synchronization patterns are less expressive than n-DRTA.

References

[AD94] R. Alur and D.L. Dill. A theory of timed automata, *Theoretical Computer Science*, 126:183-235, 1994.

[AFH94] R. Alur, L. Fix and T.A. Henzinger. A determinizable class of timed automata, in *Computer-Aided Verification*, LNCS 818:1-13, Springer Verlag, 1994.

[Con71] J. H. Conway. *Regular Algebra and Finite Machines*, Chapman and Hall, 1971.

[Dia] C. Dima. Simple Duration Calculus semantics of a real-time synchronous programming language, in manuscript, 1999.

[Di99] C. Dima. Automata and regular expressions for real-time languages, *Proceedings of the AFL'99 workshop*, Vasszeczeny, Hungary, 9-13 August 1999.

[Di00] C. Dima. Real-time automata and the Kleene algebra of sets of real numbers, in *Proceedings of STACS'2000*, LNCS series, 17-19 February 2000.

[HU] John E. Hopcroft and Jeffrey D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley/Narosa Publishing House, eighth edition, New Delhi, 1992.

[HRS98] T.A. Henzinger, J.-F. Raskin and P.-Y. Schobbens. The regular real-time languages, in *Proc. 25-th ICALP* LNCS series, Springer Verlag, 1998.