Theoretical Informatics and Applications
Theoret. Informatics Appl. **37** (2003) 159–176
DOI: 10.1051/ita:2003012

THE COMMUNICATION HIERARCHY OF TIME AND SPACE BOUNDED PARALLEL MACHINES *

Norbert Popély[1]

Abstract. We describe the communicating alternating machines and their simulation. We show that, in the case of communicating alternating machines which are bounded, simultaneously, by polynomial time and logarithmic space, the use of three communication levels instead of two does not increase computational power of communicating alternating machines. This resolves an open problem [2] concerning the exact position of machines with three communication levels in the hierarchy.

Mathematics Subject Classification. 03D15.

1. Introduction and preliminaries

Besides sequential computational models, a number of *parallel* models have been studied. According to van Emde Boas' classification [6], machine models with time complexity polynomially related to the space complexity of sequential devices form the second machine class of parallel models.

A typical example of such model is an *alternating Turing machine* [1], which is, at the same time, a generalization of nondeterminism and parallelism. This model is related to the fundamental complexity hierarchy of sequential classes by the equalities ALOGSPACE = P and APTIME = PSPACE [1].

Recall that the idea for ordinary alternating machines (ATM) is to partition the finite control states into *existential* and *universal* [1]. If, at the given moment, the program of the machine admits to execute several different instructions in the next

Keywords and phrases. Computational complexity, synchronized alternation.

* *This work was supported by the Slovak Grant Agency for Science (VEGA) under contract #1/7465/20 "Combinatorial Structures and Complexity of Algorithms".*

[1] Department of Computer Science, P. J. Šafárik University, Jesenná 5, 04154 Košice, Slovakia; e-mail: popely@mail.com

computation step, the machine *(a)* nondeterministically chooses one legal computation step, if it is in an existential state, but *(b)* follows, in parallel, all possible computation paths, if the current state is universal. By nondeterminism of (a), there may exist several different computations for the same input. By (b), the computation forks into parallel *processes*, each of them "inherits" its own private copy of all information stored on the work tapes. That is, the nondeterministically chosen computation forms a *computation tree* of parallel branches, embedded in the *full configuration tree* of all reachable configurations, with root corresponding to the initial configuration.

The notion of alternation has been generalized to a synchronized alternating machine (SATM) to study the power of communication among parallel processes of an alternating device [3, 5, 7]. Roughly speaking, the synchronized alternation captures the fact that, in a typical parallel computation, the information flows not only between parents and their offsprings, as in the case of an ordinary alternating device, but also among parallel processes active at the moment.

The synchronized alternating machines still belong to the second machine class, since SAPTIME = APTIME = PSPACE, *i.e.*, with respect to time, they are not more powerful than the ordinary alternating devices [4]. However, these two models differ significantly in space, since SALOGSPACE = PSPACE [3, 7]. Imposing both bounds simultaneously, we get SAPTIME - LOGSPACE = NP [2].

Although synchronizing alternating machines allow exchange of information between parallel working processes, the broadcasted signals does not have any structure: Each message broadcasted, by each process, is received by all others [3]. Therefore, a new model of *communicating alternating machine* [2] was established in which the processes are organized into hierarchically ordered communication groups, and information is exchanged within the given groups by local communication signals.

There was another important motivation for introducing the new machine model. Though the underlying machine for the synchronized alternation is alternating, the nature of broadcasting itself is actually *nondeterministic*. This is supported already by the simulation proving the inclusion SAPTIME $\subseteq$ APTIME [4].

Thus, it is quite natural, besides an existential communication, to consider another level of *universal* communication that does not trigger communication conflicts but rather forks the communication, if two different symbols are broadcasted. The "fork" follows the rules of universal branching, *i.e.*, both branches proceed further. The active processes running in parallel are divided into two separate communication groups; the first group consists of the processes broadcasting zeros, the second one of those broadcasting ones. From now on, the processes in either group communicate separately. Thus, the universal broadcasting of several bits prevents from communication interference between two processes having dialed different "phone" numbers.

This gives a new computational model, with the underlying machine alternating again, but the nature of broadcasting is not nondeterministic but rather alternating. More precisely, it uses two levels of communication; existential and universal.

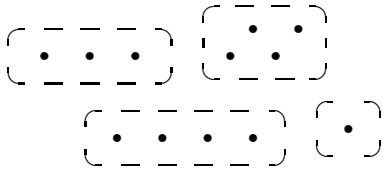

FIGURE 1. Dividing of parallel working processes (dots), into the communication groups (ovals).

The relation between the synchronized alternation and the new model of *communicating alternation* is the same as between nondeterminism and alternation.

Now we describe this new model, a communicating alternating Turing machine (CATM). In addition to the standard model, the execution of some computation steps may also depend on *communication symbols*.

The exchange of information is performed by broadcasting binary encoded messages, using a *communication alphabet* $\{0_1, 1_1, \ldots, 0_r, 1_r\}$. Symbols 0_i and 1_i are used for the ith level broadcasting, thus system has r broadcasting levels available (so called "channels"), for some $r \in \mathbb{N}$.

For better understanding we first describe the communication of $\mathrm{CA}_2\mathrm{TM}$, with the communication alphabet $\{0_1, 1_1, 0_2, 1_2\}$, using two communication levels. Figure 1 shows an example of a typical situation which may occur during the computation. The communication is restricted to the processes working in parallel within the same communication group (existential communication). The distribution into groups is organized by the processes themselves (universal communication):

- The active processes not sending any communication signals run independently in parallel.
- Each time one of the processes working in parallel tries to execute an instruction that broadcasts a communication signal it must wait until all other processes, within the same communication group, either halt or try to execute instructions that also broadcast some communication signals.
- When this happens, the process trying to execute an instruction broadcasting the existential signal 0_1 is allowed to proceed further only if all other processes, within the same group, broadcast the same signal 0_1 as well. Otherwise, the process is aborted by the *communication conflict*.
- Similarly, if the process broadcasts 1_1, all other processes must broadcast 1_1 as well.
- Finally, the processes running within the same group may also broadcast universal signals 0_2 or 1_2. In this case, the communication group is divided into two separate communication groups; the first group consists of the processes broadcasting 0_2, the second one of those broadcasting 1_2. From now on, the processes of the two offspring groups communicate separately.

Since the synchronized alternation, studied in the literature, represents ordinary alternating devices equipped with a nondeterministic communication and the new computational model, alternating machines with an alternating communication,

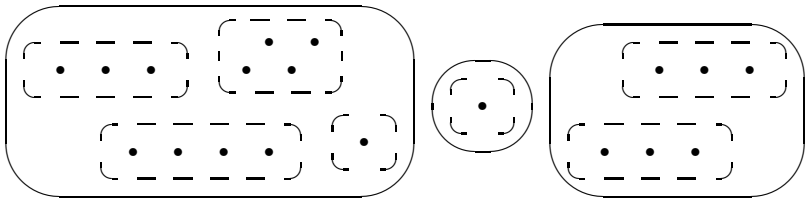

FIGURE 2. Communication groups for the level four. The first level signals are blocked by the second level boundaries (dashed ovals). These can be overcome by even higher levels.

a question arises if we can proceed further and, for example, equip alternating machines with a synchronized alternating communication. Such generalization is possible and can be iterated uniformly arbitrarily many times. This is represented by a model with r communication levels, with the existential and universal levels intertwined.

Thus, the third level communication is existential again, using the symbols 0_3, 1_3. This level allows to exchange information globally, among all processes active at the moment, across the boundaries of communication groups organized by the second level signals. That is,

- each time a process working in parallel tries to execute an instruction that broadcast a signal from $\{0_3, 1_3\}$, it must wait until all other processes, in all communication groups (organized by the second level communication), either halt or try to execute instructions that also broadcast some third level signals.
- When this happens, the process trying to broadcast 0_3 is allowed to proceed further only if all other processes, in all groups, broadcast the same signal 0_3 as well. Otherwise, the process is aborted by the communication conflict.
- Similarly, if the process broadcast 1_3, all other processes, in all groups, must broadcast 1_3 as well.

It should be obvious how the fourth level of communication is introduced. It is universal, and, similarly to the second level, it groups the third level communication into communication groups. These higher level groups are subdivided by the second level communication into nested communication subgroups of lower rank. (Fig. 2).

In general, an r-level communicating alternation machine uses the communication alphabet $\{0_1, 1_1, 0_2, 1_2, \ldots, 0_r, 1_r\}$. The existential and universal levels are intertwined, beginning with the existential pair of signals. The parallel processes are distributed into nested communication groups, the depth of nesting is $\lceil r/2 \rceil$. A signal 0_i or 1_i can be used to interact with processes within the same group of rank at most $\lceil i/2 \rceil$. An informal description of the communication is the following.

- A process trying to execute an instruction that broadcasts a communication signal of level i must wait until all other processes, within the same

group of rank $\lceil i/2 \rceil$, either halt or try to broadcast signals of level i or higher.

– When this happens, the process trying to broadcast an existential signal 0_i is allowed to proceed further only if all other processes, within the same group of rank $\lceil i/2 \rceil$, broadcast the same signal 0_i as well. The same holds, respectively, for an existential 1_i.
– If the level i is universal, the communication group of rank $\lceil i/2 \rceil$ is divided. The first and second offspring groups consist of the processes broadcasting 0_i and 1_i, respectively. In addition, possible idle processes (in any), waiting for a communication by higher level signals, are expelled to a separate offspring group, where they wait for their turns. The group division applies to all nested subgroups of lower ranks as well.

Note that there can exist some processes broadcasting on different levels at the same time within the same group. The situation for the existential level is obvious: Once there is a process broadcasting, say 1_i, all other interacting processes must broadcast 1_i or else a conflict arises. Since the universal level does not trigger conflicts but rather forks communication, and since a higher level communication should not block the lower levels but rather be delayed, a higher level process is not "aborted", as in the existential case, but rather "expelled". This interference about processes communicating on different levels can be avoided, as presented in Lemma 2.1 (for details, see [2, Lem. 4.2]).

We can now present a formal definition, introduced in [2].

Definition 1.1. A *program* of k-tape r-level communicating alternating Turing machine ($\mathrm{CA}_r\mathrm{TM}$) is a set $I \subseteq Q \times \Sigma \times \Delta^k \times \Delta^k \times \{-1, 0, 1\}^{k+1} \times Q \times \{\varepsilon, 0_1, 1_1, \ldots, 0_r, 1_r\}$, where $Q = E \cup U$ is a finite set of states, consisting of two disjoint subsets of existential an universal states, Σ is a finite input alphabet and Δ is a finite work tape alphabet.

An *instruction* $< q;\ a;\ b_1, \ldots, b_k;\ b'_1, \ldots, b'_k;\ d;\ d_1, \ldots, d_k;\ q', c > \ \in I$ is interpreted as follows: if the machine is in the state q and scans the symbol a from the input tape and the symbols $b_1, \ldots, b_k$ from the work tapes, then it rewrites $\{b_1, \ldots, b_k\}$ by $b'_1, \ldots, b'_k$, respectively, moves the corresponding tape heads in the directions $d;\ d_1, \ldots, d_k$ and change its state from q to q', broadcasting the communication signal c. (If $c = \varepsilon$, no signal is broadcasted). Such action is called a *computation step*.

A *configuration* is an element of $Q \times \mathbb{N} \times (\Delta^* \times \mathbb{N})^k$, representing the (current) finite control state, the non blank contents of all tapes, together with all tape head positions.

Definition 1.2. A *full configuration tree* of the machine for an input w is a (possibly infinite) tree the nodes of which represent configurations, with root corresponding to the initial configuration. The edges correspond to single computation steps and are labeled by the broadcasted communication signals $\{0_1, 1_1, \ldots, 0_r, 1_r\}$, with no labels for transitions not broadcasting any signals.

The *communication history* of a node v in a full configuration tree T is the sequence of edge labels (communication symbols) along the path from the root to v. It will be denoted by $h_1(v)$.

For $i \in \{1, \ldots, r+1\}$, an ith *level of the communication history* of a node v is the string $h_i(v)$, obtained from $h_1(v)$ by a homomorphism that erases all lower level communication, *i.e.*, that maps $0_k \mapsto \varepsilon$ and $1_k \mapsto \varepsilon$, for $k < i$, but $0_k \mapsto 0_k$ and $1_k \mapsto 1_k$, for $k \geq i$.

A *computation tree* is a subtree T' of the full configuration tree T satisfying the following:

(a) A node v is in T' only if it is a root or if its father is also in T'.

(b) If a node v in T' labeled by an existential configuration has a son which is also in T', then no other son of v in T is in T'.

(c) If a node v in T' labeled by a universal configuration has a son which is also in T', then all other sons of v in T are in T' as well.

(d) If, for any odd $i \leq r$, and any pair of nodes u and v in T',

$$|h_i(u)| = |h_i(v)|, \quad \text{but} \quad h_i(u) \neq h_i(v),$$

then, for some nodes α_u and α_v along the paths from the root to u and v, respectively,

$$|h_{i+1}(\alpha_u)| = |h_{i+1}(\alpha_v)|, \quad \text{but} \quad h_{i+1}(\alpha_u) \neq h_{i+1}(\alpha_v).$$

(e) T' is maximal with respect to (*a*)–(*d*), *i.e.*, adding any additional node of T to T' violates at least one of the above conditions.

The input w *is accepted*, if there exists a computation tree such that all its branches are terminated in accepting configurations.

Note that a higher rank division applies to all nested subgroups of lower ranks as well, since $h_i(x) \neq h_i(y)$ implies $h_{i-1}(x) \neq h_{i-1}(y)$. The reader may easily verify that all other details correspond to the informal presentation above.

The above computational model is a natural generalization of a synchronized alternating machine (SATM) [3, 5, 7], which in turn is a generalization of the standard alternating machines (ATM) [1].

Definition 1.3. A *synchronized alternating machine* is a communicating alternation machine using only the existential communication signals 0_1 and 1_1.

A (standard) *alternating machine* is a communicating alternation machine using no communication signals, *i.e.*, the empty communication alphabet, with $r = 0$.

Definition 1.4. The machine is $t(n)$ *time bounded*, if, for each input of length n, no computation tree has a branch longer than $t(n)$ computation steps.

The machine is $s(n)$ *space bounded*, if, for each input of length n, no computation tree has a node corresponding to a configuration that uses more than $s(n)$ space on any of its work tapes (not taking into account the length of the read-only input tape).

The classes of languages recognizable by communicating alternation machines in $O(t(n))$ time or $O(s(n))$ space with r communication levels will be denoted by $\mathrm{CA}_r\mathrm{TIME}(t(n))$ or $\mathrm{CA}_r\mathrm{SPACE}(s(n))$, respectively. The class of languages recognizable by such machines in $O(t(n))$ time and $O(s(n))$ space simultaneously will be denoted by $\mathrm{CA}_r\mathrm{TIME}\text{-}\mathrm{SPACE}(t(n), s(n))$, $\mathrm{CA}_r\mathrm{PTIME}\text{-}\mathrm{LOGSPACE}$ is a shorthand notation for classes of languages recognized by such machines in polynomial time an logarithmic space.

$\mathrm{SATIME}(t(n)) = \mathrm{CA}_1\mathrm{TIME}(t(n))$ and $\mathrm{ATIME}(t(n)) = \mathrm{CA}_0\mathrm{TIME}(t(n))$ denote the classes of languages recognizable by $O(t(n))$ time bounded synchronized alternating and alternating machines, respectively. The corresponding notation $X\mathrm{SPACE}(s(n))$, $X\mathrm{TIME}\text{-}\mathrm{SPACE}(t(n), s(n))$, and $X\mathrm{PTIME}\text{-}\mathrm{LOGSPACE}$, for $X \in \{\mathrm{SA}, \mathrm{A}\}$, will be used for space and time-and-space bounded machines.

Now we show the relation between time bounded alternating and synchronized alternating machines. This result was originally proved in [7], but we will investigate the size of the used space, too.

Theorem 1.5. *For each function $t(n)$, constructible in time $O(t(n))$ and in space $O(t(n))$, and for each function $s(n)$,*

$$\mathrm{SATIME}\text{-}\mathrm{SPACE}(t(n), s(n)) \subseteq \mathrm{ATIME}\text{-}\mathrm{SPACE}(t(n), s(n) + t(n)).$$

Proof. (Sketch) Let $\mathcal{L}(\mathcal{M}) \in \mathrm{SATIME}\text{-}\mathrm{SPACE}(t(n), s(n))$. Now we describe simulation of machine $\mathcal{M}$ by $\mathcal{M}'$, where $\mathcal{L}(\mathcal{M}') \in \mathrm{ATIME}\text{-}\mathrm{SPACE}(t(n), s(n) + t(n))$:

The machine $\mathcal{M}'$ has an auxiliary pushdown store and one more work tape. The initial process, on the auxiliary work tape, deterministically computes the value of the function $t(n)$. This is possible in time $O(t(n))$ and space $O(t(n))$, by the assumption. After computing the value of $t(n)$, $\mathcal{M}'$ nondeterministically generates the number of broadcasted synchronizing signals t', for some nondeterministically chosen $t' \leq t(n)$, in time $O(\log t(n))$. After that, $\mathcal{M}'$ generates a synchronizing sequence of length t' to the auxiliary pushdown store. This can be done in time $O(t(n))$ and in space $O(t(n))$. Then $\mathcal{M}'$ simulates $\mathcal{M}$ as follows:

- If the machine $\mathcal{M}$ does not communicate, then $\mathcal{M}'$ executes the same instructions which were executed by a machine $\mathcal{M}$, including corresponding existential/universal branchings. (If the branching is universal, then all descendants of the process inherit the same content of the pushdown store.)
- If some process x of $\mathcal{M}$ tries to communicate, broadcasting some synchronizing signal c, then:
  - If the auxiliary pushdown store is not empty, then the corresponding process x' of $\mathcal{M}'$ pops one pushdown symbol from top and compares it with the synchronizing signal c. If they are equal, then x' continues on the simulation. Otherwise (in case they are different) x' rejects the input (HALT for wrong synchronizing sequence).
  - If the auxiliary pushdown store of x' is empty, then x' rejects the input (HALT for wrong number of synchronizing signals).

In the case the machine $\mathcal{M}$ accepts the input, the machine $\mathcal{M}'$ accepts, because there exists a sequence of synchronizing signals of length $t' \leq t(n)$, which is broadcasted by each process of $\mathcal{M}$, such that each process of $\mathcal{M}$ stops in an accepting configuration. For this nondeterministically generated synchronizing sequence in the auxiliary pushdown store, all processes x' of $\mathcal{M}'$ finish in accepting configurations, for the right sequence of nondeterministic choices.

In the case the machine $\mathcal{M}$ rejects the input, then for each sequence of nondeterministic choices, each process x of $\mathcal{M}$:

- is either aborted by a communication conflict, with some other processes trying to broadcast a different synchronizing signals at the same time, (in this case the corresponding process x' of $\mathcal{M}'$ must stop and reject because its "broadcasted" signal and the top of the pushdown store are different),
- or, without a communication conflict, x stops in a rejecting configuration. (In this case the corresponding process x' of $\mathcal{M}'$ stops and rejects, too.)

Thus the machine $\mathcal{M}'$ accepts the input if an only if $\mathcal{M}$ accepts. In addition, $\mathcal{M}'$ works in time $O(t(n))$ and in space $O(s(n) + t(n))$. Since $\mathcal{M}'$ does not broadcasts any synchronizing signals (the broadcasting is simulated by the pushdown store), $\mathcal{M}'$ is a classical alternating machine. □

We see that the synchronizing alternating machines are not more powerful than the classical alternating machines, with respect to time. However, there is the fundamental difference in the used memory.

Quite surprisingly, using more communication levels than two does not increase the power of time bounded machines; for each $r \geq 2$, $\mathrm{CA}_r\mathrm{PTIME} = \mathrm{CA}_2\mathrm{PTIME} = \mathrm{NEXPTIME}$. The situation changes very dramatically for space bounds. The communication hierarchy of space bounded CA_r-machines is infinite, since, for each $r \in \mathbb{N}$, $\mathrm{CA}_{r+2}\mathrm{LOGSPACE} = \mathrm{CA}_r\mathrm{PSPACE}$. Thus, $\mathrm{CA}_{2r}\mathrm{LOGSPACE} = \mathrm{D}(\mathrm{EXP})^r\mathrm{TIME}$ and $\mathrm{CA}_{2r+1}\mathrm{LOGSPACE} = (\mathrm{EXP})^r\mathrm{SPACE}$, for each $r \geq 0$. (Here "$(\mathrm{EXP})^r$" represents an r times iterated exponential function. For example, EXPEXPSPACE is denoted by $(\mathrm{EXP})^2\mathrm{SPACE}$.) This indicates that we can solve very complex problems on parallel devices keeping the size of elements very "small", but the possibility to speed up the computation time is limited, even if a parallel system can grow exponentially and reconfigure its topology during the computation.

This extends the well-known characterization of DLOGSPACE, NLOGSPACE, P, and PSPACE by deterministic, nondeterministic, alternating, and synchronized alternating two-way multihead finite automata. The above characterization represents just the first four members of an infinite hierarchy of multihead automata, representing the entire fundamental complexity hierarchy. The first "new" class added, is DEXPTIME, corresponding to the two-level communicating alternation [2].

Since already the communication hierarchy of time bounded machines is finite, it is not very surprising that the same holds for machines with simultaneous bounds

on both time and space. However, this hierarchy is not equal to a time hierarchy. The following corollary summarizes the known results [2]:

Corollary 1.6.

$$\begin{array}{lcll}
\text{DLOGSPACE} & = & \text{DPTIME - LOGSPACE}, & \\
\text{NLOGSPACE} & = & \text{NPTIME - LOGSPACE}, & \\
\text{P} & = & \text{APTIME - LOGSPACE} & (\text{CA}_0), \\
\text{NP} & = & \text{SAPTIME - LOGSPACE} & (\text{CA}_1), \\
\text{PSPACE} & = & \text{CA}_2\text{PTIME - LOGSPACE}, & \\
? & = & \text{CA}_3\text{PTIME - LOGSPACE}, & \\
\text{NEXPTIME} & = & \text{CA}_r\text{PTIME - LOGSPACE}, & \text{for each } r \geq 4.
\end{array}$$

(Formally, nondeterministic and deterministic machines may be viewed as the minus first and minus second levels of the hierarchy.)

The exact characterization of the class CA_3PTIME - LOGSPACE in the hierarchy was the open problem. In the following section, we shall give the exact position of this class in the hierarchy, by showing that it is equivalent to CA_2PTIME - LOGSPACE, which in turn is equal to PSPACE. Thus, adding the third, global, level of communication to machines with two levels, that can exchange information locally within communication groups, does not increase the computational power, if the machines are simultaneously bounded by polynomial time and logarithmic space.

2. The simulation of CA_3 TM by CA_2 TM

As an initial point of this simulation, the construction presented in Theorem 1.5 is used (originally published in [7]). It describes a simulation of a synchronized alternating $t(n)$ time bounded machine by an equivalent $O(t(n))$ time bounded machine without synchronization. The construction is based on the use of an auxiliary pushdown store, in which the alternating machine nondeterministically generates the entire sequence of synchronization signals of length $O(t(n))$ in the initial phase. Further, the simulation proceeds "straightforwardly" but one difference; every time the simulating machine must execute a computation step in which the process of the original machine tries to communicate with other processes, the simulating machine pops the corresponding sequence of signals from the additional pushdown store. Since the pushdown store was filled in the initial existential phase, all processes running in parallel "inherit" its content, and hence all processes proceed according to the same sequence of signals. Thus, the need of a communication is avoided.

Our simulation of three communication levels by two is based on the same idea: In the initial phase, a sequence of third level communication signals is nondeterministically generated, and it is "inherited" in the computation that follows. Thus, there is no need to use more than two levels of communication. However, the whole construction is much more complex than that in [7] for the following

reasons: (a) To store the third level communication history, we can not use a single pushdown store (neither some other sequential memory as, for example, tape), because this increases the used memory from $s(n)$ to $O(s(n) + t(n))$. Therefore, the information must be distributed over the blocks, stored in $O(t(n))$ parallel auxiliary processes. (b) To "reconstruct" the information about the third level communication from these separate $O(t(n))$ processes, active processes exchange information using only the first two communication levels. However, the simulated original machine also uses the first two levels of communication. To avoid an interference between the original messages broadcasted on the first two levels and the imitated third level, we have to encode all signals by a suitable binary code and to alternate transmission between the "original" first two and the third levels.

Before passing further, we need to avoid some technical problems by putting the original machine into a "normal form": we first avoid an interference among processes broadcasting at different communication levels, at the same time. This normal form was proved in [2].

Lemma 2.1. *(First normal form): For each* CA_r*-machine* A*, there exists an equivalent machine* A' *such that no pair of processes will ever interact using different communication levels. In addition, a process in a configuration broadcasting a communication signal has a unique direct descendant.*

Next, we show how we can construct a machine that broadcasts always exactly $2^{\lceil \log t(n) \rceil}$ third level communication signals, instead of some $t' \leq t(n)$.

Lemma 2.2. *(Second normal form): Let the function* $t(n)$ *is constructible in time* $O(t(n))$ *and in space* $O(\log t(n))$*. Then, for each function* $s(n)$ *and for each machine* $\mathcal{M}$*, where* $\mathcal{L}(\mathcal{M}) \in \mathrm{CA_3TIME\text{-}SPACE}(t(n), s(n))$*, there exist an equivalent machine* $\mathcal{M}'$*, such that* $\mathcal{L}(\mathcal{M}') \in \mathrm{CA_3TIME\text{-}SPACE}(t(n), s(n) + \log t(n))$ *and* $\mathcal{M}'$ *broadcasts exactly* $2^{\lceil \log t(n) \rceil}$ *third level communication signals, for each input of the length* n*, and by each process running in parallel.*

Proof. We describe the simulation of the original machine $\mathcal{M}$ by the machine $\mathcal{M}'$ with the above properties. Initially, $\mathcal{M}'$ deterministically computes the value of $t(n)$, on an additional auxiliary work tape. This is possible in time $O(t(n))$ and in space $O(\log t(n))$. To store value of the function $t(n)$, $\mathcal{M}'$ uses $\lceil \log t(n) \rceil$ bits. After computing $t(n)$, $\mathcal{M}'$ sets all used bits to 1. Thus, we have stored a binary code of the number $2^{\lfloor \log t(n) \rfloor + 1} - 1$. Then $\mathcal{M}'$ simulates the machine $\mathcal{M}$ but, every time some process x of $\mathcal{M}$ broadcasts a third level communication signal $c_3 \in \{0_3, 1_3\}$, the corresponding process x' of $\mathcal{M}'$ first decrements a value of the counter containing initially $2^{\lceil \log t(n) \rceil}$ on the auxiliary work tape, and then it simulates the corresponding computation step of $\mathcal{M}$. If some process x of $\mathcal{M}$ accepts the input, then:

- If the counter for the third level signals is empty (*i.e.* it contains only zero), then the corresponding process x' of $\mathcal{M}'$ accepts the input.
- If the counter is not empty, then x' repeatly broadcasts existentially (nondeterministically) chosen communication signals 0_3 or 1_3 and it decrements the value in the counter until its value is zero.

Hence, the whole computation runs on $\mathcal{M}'$ in the same way as on $\mathcal{M}$, with the following difference: if, in the original machine $\mathcal{M}$, processes broadcasted k third level communication signals, where $k < 2^{\lceil \log t(n) \rceil}$, then the original sequence of the third level signals will be completed with some signals to the length $2^{\lceil \log t(n) \rceil}$. Thus, $\mathcal{M}'$ works in time $O(t(n))$ and space $O(s(n) + \log t(n))$. □

Now we are ready for the main theorem.

Theorem 2.3. *For each function $t(n)$, constructible in time $O(t(n))$ and in space $O(\log t(n))$, and for each function $s(n)$,*

$\mathrm{CA_3TIME\text{-}SPACE}(t(n), s(n)) \subseteq \mathrm{CA_2TIME\text{-}SPACE}(t(n), s(n) + \log t(n))$.

Proof. Let language $\mathcal{L}(\mathcal{M}_1) \in \mathrm{CA_3TIME\text{-}SPACE}(t(n), s(n))$. By Lemma 2.2 (second normal form) we can replace $\mathcal{M}_1$ by a machine $\mathcal{M}_2$, where $\mathcal{L}(\mathcal{M}_2) \in \mathrm{CA_3TIME\text{-}SPACE}(t(n), s(n) + \log t(n))$, such that machine $\mathcal{M}_2$ broadcasts exactly $2^{\lceil \log t(n) \rceil}$ third level communication signals. Using Lemma 2.1 (first normal form), it is possible to replace $\mathcal{M}_2$ by a machine $\mathcal{M}$, such that $\mathcal{L}(\mathcal{M}) \in \mathrm{CA_3TIME\text{-}SPACE}(t(n), s(n) + \log t(n))$ and no pair of processes of machine $\mathcal{M}$ will ever interact using different communication levels, and broadcasting exactly $2 \cdot 2^{\lceil \log t(n) \rceil}$ third level communication signals. (From the simulation of Lem. 2.1 (first normal form), it follows that each third level signal $c_3 \in \{0, 1\}$ is encoded by the sequence of communication signals $1_1 1_2 0_3 c_3$, *i.e.*, the number of broadcasted third level communication signals will *exactly* duplicate.) Moreover, each process of the machine $\mathcal{M}$, which is in a broadcasting state and broadcasts a communication signal, has exactly one direct descendant. Now, we describe simulation of $\mathcal{M}$ by a machine $\mathcal{M}'$, where $\mathcal{L}(\mathcal{M}') \in \mathrm{CA_2TIME\text{-}SPACE}(t(n), s(n) + \log t(n))$:

First, the initial process of $\mathcal{M}'$ is universally branched into two processes x and y. The process x and all its descendants provide the simulation of the machine $\mathcal{M}$. (These processes will be called *X-processes*). The process y and all its descendants, which are called *Y-processes*, first generate, nondeterministically, a sequence of third level communication signals, and keep this sequence distributed in their finite control states. Every time when *X-processes* need to communicate on the third communication level, *Y-processes* will pass the corresponding third level signal to *X-processes*, suitably encoded by a message in the first two levels of communication.

Taking into account that the sequence of third level signals is generated in the initial phase, before any splitting of processes into communications groups, all third level signals broadcasted by *Y-processes* will be same in every communications group. This can be done because, before the splitting some communication group into two communication subgroups, each of *Y-processes* splits itself into the two identical descendants so that third level communication information comes into each subgroup.

In the initial phase, the initial *Y-process* on the first track of its auxiliary work tape deterministically computes the value of $t(n)$ (using some additional work tapes, if necessary). Because the number of broadcasted third level signals will be $2 \cdot 2^{\lceil \log t(n) \rceil}$, storage of the whole sequence by one process would have required

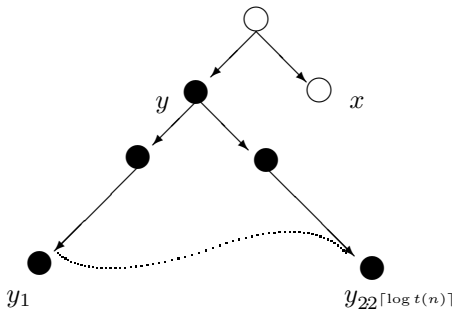

FIGURE 3. *Y-processes*, initial branching.

too big space (more precisely $2{\cdot}2^{\lceil \log t(n) \rceil} \in \Theta(t(n))$ bits). Therefore, the initial *Y-process* universally splits $1 + \lceil \log t(n) \rceil$-times, which creates $2{\cdot}2^{\lceil \log t(n) \rceil}$ processes working in parallel (Fig. 3). Each time the process branches universally, it stores a bit 0 or 1 to the second track of its work tape. Hence, after $1 + \lceil \log t(n) \rceil$ branchings, $2{\cdot}2^{\lceil \log t(n) \rceil}$ parallel working *Y-processes* will arise from the initial *Y-process*, and each of them has the value $t(n)$ on the fist track of its work tape and one binary encoded number $i \in \{0, 1, 2, \ldots, 2{\cdot}2^{\lceil \log t(n) \rceil} - 1\}$ on the second track. These numbers correspond to the chosen computational way in the computation tree. Then each *Y-process* increments the second track value by 1. Hence, particular *Y-processes* are assigned by values $i \in \{1, 2, 3, \ldots, 2{\cdot}2^{\lceil \log t(n) \rceil}\}$.

$\vdash$	1	...	1	1	1	1	1	$\sharp$
$\vdash$	0	...	1	1	0	1	1	$\sharp$

$t(n)$
$i \in \{1, \ldots, 2{\cdot}2^{\lceil \log t(n) \rceil}\}$

Finally, each *Y-process* nondeterministically chooses one bit – a third level communication signal – and stores it into the control state. Thereby, we have nondeterministically generated a sequence of the third level signals of the original machine $\mathcal{M}$. This procedure requires time $O(\log t(n))$ and no *Y-process* uses space greater than $O(\log t(n))$, since each process remembers only one communication signal and its position in the sequence (*i.e.*, some $i \in \{1, 2, 3, \ldots, 2{\cdot}2^{\lceil \log t(n) \rceil}\}$), which is binary coded. Then the machine $\mathcal{M}'$ simulates $\mathcal{M}$ with the following differences:

Whenever processes of $\mathcal{M}$ do not communicate, the corresponding *X-processes* of $\mathcal{M}'$ execute the same instructions, and *Y-processes* of $\mathcal{M}'$ are in the waiting state. If some process x of $\mathcal{M}$ broadcasts a communication signals $w \in \{0_1, 1_1, 0_2, 1_2, 0_3, 1_3\}$, then the corresponding process x' of $\mathcal{M}'$ broadcasts signals encoded as follows:

$\mathbf{x_k}$	0_1	1_1	0_2	1_2	0_3	1_3
$\mathbf{x'_k}$	$0_1 1_1 0_1$	$0_1 1_1 1_1$	$1_1 0_1 0_2$	$1_1 0_1 1_2$	$1_1 1_1 0_1$	$1_1 1_1 1_1$

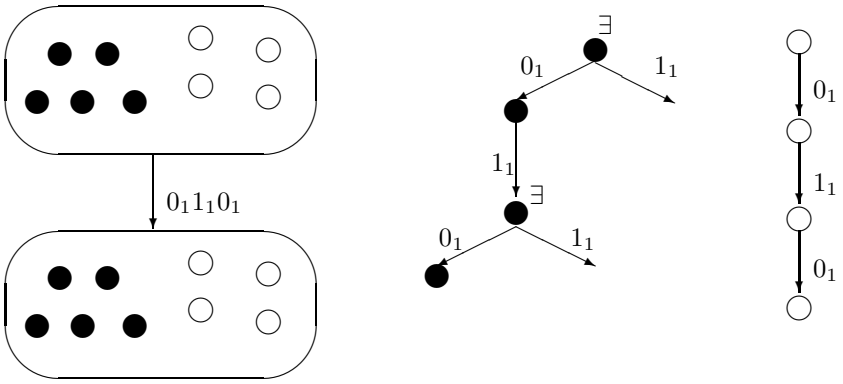

FIGURE 4. First level communication (for $c_1 = 0_1$). Full circles represent *Y-processes*, empty cycles *X-processes*.

In every case, before the signal w itself, it broadcasts the encoded number of communication level using the first level signals (*i.e.*, $0_1 1_1$, $1_1 0_1$, or $1_1 1_1$). This information is unnecessary for the *X-processes*, but, on its base, *Y-processes* will "know", whether they have to ignore the third broadcasted bit in waiting state (for the communication on the first and the second levels), or to supply the third bit to *X-processes* from the initially guessed third level communication history. More detailed description of particular cases is the following.

– If some process x broadcasts a communication signal $c_1 \in \{0_1, 1_1\}$, then the corresponding process x' of machine $\mathcal{M}'$ first broadcasts the binary code of the communication level (*i.e.*, $0_1 1_1$) and after it, it broadcasts the signal c_1. If some other processes within the same communication group would try to broadcast communication signals of different level, second or third, then a communication conflict would occur, because these processes in $\mathcal{M}'$ encode their communication signals $b \in \{0_2, 1_2\}$ or $b \in \{0_3, 1_3\}$ by $1_1 0_1 b$ (or, respectively, $1_1 1_1 b \ldots$). If some processes, within the same group, would try to broadcast signals of the same level (*i.e.*, first), but different from c_1, we would also have the communication conflict, since these signals are encoded by $0_1 1_1 c_1'$, with $c_1' \neq c_1$. In any case, the computation will not be aborted only if all other *X-processes*, within the same communication group, broadcast the same signal c_1 as well.

  Concerning *Y-processes* of $\mathcal{M}'$ present in the same communication group, they all branch existentially in the next two steps, broadcasting two nondeterministically chosen bits from $\{0_1, 1_1\}$. Since we have an *X-processes* broadcasting $0_1 1_1 c_1$, only the branches of *Y-processes*, that broadcast $0_1 1_1$ will proceed further. As *Y-processes* were forced to proceed along the branch broadcasting the sequence $0_1 1_1$, they all "know" that they must ignore the third broadcasted signal. Thus each *Y-process* is branched once more, existentially for the third time, such that signals

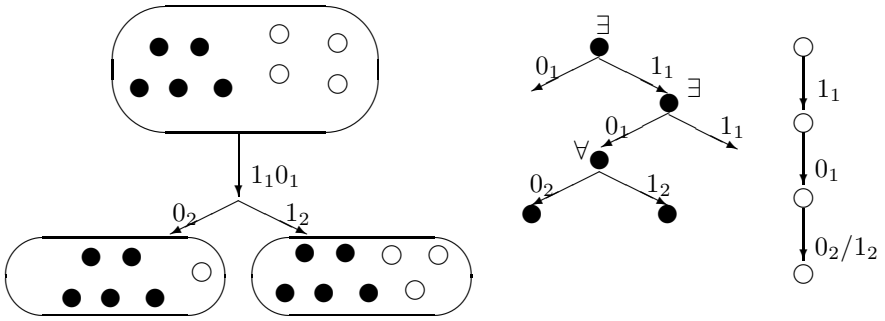

FIGURE 5. Second level communication – splitting the group.

0_1 or 1_1 are broadcasted. This allows them to skip and ignore the broadcasted signal c_1. As a result, *Y-processes* ignore the communication signal on the first level, waiting for the next communication (Fig. 4).

– If some processes broadcast communication signals $c_2 \in \{0_2, 1_2\}$, then the corresponding processes of $\mathcal{M}'$ first broadcast the binary code of the communication level (*i.e.*, $1_1 0_1$) and, after it, they broadcast their signals c_2. If some processes within the same communication group would try to broadcast communication signal of different level, first or third, then a communication conflict would occur, because these processes encode their signals $b \in \{0_1, 1_1\}$ or $b \in \{0_3, 1_3\}$ by $0_1 1_1 b$ (or, respectively, $1_1 1_1 b \ldots$). If all processes within the group broadcast the second level signals $c_2 \in \{0_2, 1_2\}$, the corresponding *X-processes* of $\mathcal{M}'$ broadcast messages of the form $1_1 0_1 c_2$. As a result, no conflict arises in the first two steps, but, in the third step, the group is divided into two separate communication groups, the first communication group consists of all *X-processes* broadcasting $c_2 = 0_2$, the second one of those broadcasting $c_2 = 1_2$. From then on, *X-processes* within the separate offspring groups communicate separately by the first level signals.
Concerning *Y-processes* in the group, since they nondeterministically broadcast the two bits from $\{0_1, 1_1\}$, they are forced to follow the path along $1_1 0_1$ in this case. Hence, they "know" that the next signal belongs to the second level. Therefore, in third step, *Y-processes* split *universally*, broadcasting 0_2 and 1_2. That ensures that *both* offspring communication groups will contain all *Y-processes* after the splitting (Fig. 5).

– If some process x broadcasts a communication signal $c_3 \in \{0_3, 1_3\}$, then the corresponding process x' of $\mathcal{M}'$ first broadcasts the binary code of the communication level (*i.e.*, $1_1 1_1$) and, after it, it broadcasts the corresponding first level signal $c_1 \in \{0_1, 1_1\}$. If some processes within the same communication group would try to broadcast communication signals of different level, first or second, then a communication conflict would

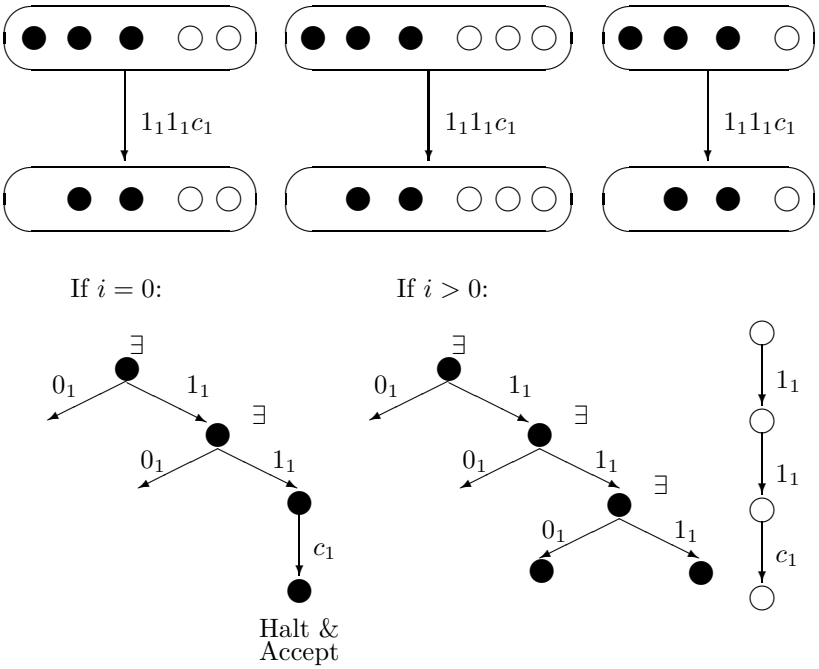

FIGURE 6. Third level communication. Ovals represent particular communication groups.

occur, because these processes in $\mathcal{M}'$ encode their signals $b \in \{0_1, 1_1\}$ or $b \in \{0_2, 1_2\}$ by $0_1 1_1 b$ (or, respectively, $1_1 0_1 b$).

Similarly as in previous cases, *Y-processes* are, due to existentially broadcasted signals 0_1 and 1_1, forced to pass along the path $1_1 1_1$ in the first two steps. Thereby, all *Y-processes* will "know" that the third level broadcasting of $\mathcal{M}$ is simulated. According to this, each *Y-process* of $\mathcal{M}'$ decrements the value of its counter i on its work tape. Recall that, in the initial phase, *Y-processes* were numbered by values $i \in \{1, \ldots, 2 \cdot 2^{\lceil \log t(n) \rceil}\}$, where *Y-process* with number i stores the ith third level communication bit in its control state. (After the broadcasting of the first k third level signals has been simulated, the first k *Y-processes* (in each communication group) are already terminated in a halting and accepting state, the remaining *Y-processes* are numbered by values i decremented by k.) After decrementing of its counter i, each *Y-process* is testing whether $i = 0$.

– If $i = 0$ (Fig. 6), then the *Y-process* deterministically broadcasts its stored third level communication bit $c \in \{0, 1\}$, encoded by $\{0_1, 1_1\}$, and stops in an accepting state. Thereby, it is guaranteed that each *X-process* will "survive" the third step of its broadcasting $1_1 1_1 c_1$, only if c_1 is identical with the bit broadcasted by the ith *Y-process*. (Otherwise, a communication conflict occurs and the whole computation

is halted.) Since, in all communication groups, we have the same *Y-processes* after the branching, all *X-processes* in all communication groups must simulate the broadcasting of the same sequence of the third level communication signals. (The sequence of the third level signals — of length $2{\cdot}2^{\lceil \log t(n) \rceil}$ — was generated by *Y-processes* in the initial phase, before any splitting into groups.)

– If $i > 0$, then the ith *Y-process* knows that the time to broadcast "its" stored signal has not come yet. Hence, it "ignores" the third level signal of the encoded message $1_1 1_1 c_1$, *i.e.*, it branches existentially, where the particular branches broadcast signals 0_1 and 1_1. (Fig. 6.)

For the simulation of the machine $\mathcal{M}$, *X-processes* do not need time longer than $O(t(n))$ and space bigger than $O(s(n))$. The only slow-down of *X-processes* is due to the encoding of each communication signal by three signals. *Y-processes* need, in addition, to decrement gradually the value of their counters from i to zero using the operation $i \leftarrow i-1$. Since the value of each counter i is not greater than $2{\cdot}2^{\lceil \log t(n) \rceil}$, no *Y-process* neither uses time longer than $O(t(n))$, nor space bigger than $O(\log t(n))$.

Thus, the machine $\mathcal{M}'$ works in time $O(t(n))$ and in space $O(s(n) + \log t(n))$. □

Since CA_2-machines are special cases of CA_3-machines, we get:

Corollary 2.4. *For each function $t(n)$, constructible in time $O(t(n))$ and in space $O(\log t(n))$, and for each function $s(n) \geq \log t(n)$,*

$$\mathrm{CA}_3\mathrm{TIME}\text{-}\mathrm{SPACE}(t(n), s(n)) = \mathrm{CA}_2\mathrm{TIME}\text{-}\mathrm{SPACE}(t(n), s(n)).$$

3. Conclusions

It should be obvious that polynomials (*i.e.*, functions $t(n) = n^k$, for some $k \geq 1$) can be constructed in polynomial time and logarithmic space, simultaneously. This gives:

Corollary 3.1. $\mathrm{CA}_3\mathrm{PTIME}\text{-}\mathrm{LOGSPACE} = \mathrm{CA}_2\mathrm{PTIME}\text{-}\mathrm{LOGSPACE}$.

From Corollary 3.1, we obtain the complete characterization of hierarchy of communicating alternating machines, which are both time and space bounded. This completes the table presented in Corollary 1.6:

DLOGSPACE	=	DPTIME - LOGSPACE	(CA_{-2}),
NLOGSPACE	=	NPTIME - LOGSPACE	(CA_{-1}),
P	=	APTIME - LOGSPACE	(CA_0),
NP	=	SAPTIME - LOGSPACE	(CA_1),
PSPACE	=	CA_2PTIME - LOGSPACE,	
PSPACE	=	CA_3PTIME - LOGSPACE,	
NEXPTIME	=	CA_rPTIME - LOGSPACE,	for each $r \geq 4$.

In other words, the standard alternating machines with polynomial time bounds and simultaneous logarithmic space bounds represent an alternative characterization of the class P. Adding the power of communication among processes running in parallel (one level of communication) raises the power of such machines from P to NP. By allowing, in addition, a second universal level of communication, that enables the local exchange of information within dynamically organized communication groups, makes the class of such machines equivalent to PSPACE. The contribution of this paper is that adding the third level of communication, that provides, besides the local exchange of information within the groups, also the global communication among all active processes, does not increase the computation power. Such machines still correspond to PSPACE. This resolves an open problem stated in [2]. Adding one more level, the fourth, that groups the third level communication into groups of higher rank (containing nested groups for the first level communication), shifts the power from PSPACE to NEXPTIME. Finally, any additional number of levels can be simulated by four levels.

Open problem: Does Corollary 2.4 hold also in the case of $s(n) \not\geq \log t(n)$, or if $t(n)$ is not constructible?

It is possible to discard the assumption about constructibility of $t(n)$, if we consider the weak complexity measures. The machine is *weakly* $t(n)$ time bounded if, for each *accepted* input of length n, there exists *at least one* accepting computation tree the branches of which never execute more than $t(n)$ computation steps. Compare with Definition 1.4, where it is required that *no* computation uses more time than $t(n)$, no matter whether the input is accepted or rejected. The same difference is between the strong (classical) and weak definition of space complexity. We shall denote the corresponding weak complexity classes by adding the prefix "weak–".

Theorem 3.2. *For all functions $t(n)$ and $s(n)$:*

$$\mathsf{weak}\text{–}\mathrm{CA}_3\mathrm{TIME}\,\text{-}\,\mathrm{SPACE}(t(n), s(n)) \subseteq \mathsf{weak}\text{–}\mathrm{CA}_2\mathrm{TIME}\,\text{-}\,\mathrm{SPACE}(t(n), s(n) + \log t(n)).$$

Proof. We can use the simulation described in Theorem 2.3. However, if we want to leave out the assumption about constructibility, we need to do some modifications.

Instead of computing $t(n)$ on auxiliary work tape, the initial process of $\mathcal{M}'$ nondeterministically generates the value $t' = 2^p$, where $p \in \mathbb{N}$. For computing of this value, time $O(\log t')$ and space $O(\log t')$ is sufficient. Then $\mathcal{M}'$ works in the same way as in Theorem 2.3, *i.e.*, the initial process is universally branched into two processes x and y. Descendants of the process x provide the simulation of $\mathcal{M}$ and descendants of y the simulation of the third level communication.

Since the original machine $\mathcal{M}$ is weakly $t(n)$ time bounded, for each accepted input of length n there exists at least one accepting computation, whose time does not exceed some value $t'' \leq t(n)$. But then there exists $t' = 2^p$ (the power of two), satisfying $t'' \leq t' \leq 2t'' \leq 2t(n)$, such that $\mathcal{M}$ has an accepting computation whose time does not exceed the value of t'. (Among others, the number of broadcasted

third level signals does not exceed t'.) For t' nondeterministically chosen in such a way (and a "correct" sequence of additional nondeterministic decisions), $\mathcal{M}'$ will have accepting computation, too, and its time does not exceed the value $O(t(n))$. It should also be obvious that *X-processes* do not use more space than $O(s(n))$, the space used by processes of the original machine $\mathcal{M}$. The space used by *Y-processes* is, on the other hand, bounded by $O(\log t') \leq O(\log t(n))$.

Thus, for weak time bound, we do not need any assumption about constructibility of $t(n)$. The disadvantage of this machine is that the machine has, besides the computation of length t', many additional computations (according to "incorrectly" generated values of t' in the initial phase), with time greater than $t(n)$. One branch is even infinite (permanently increasing the value of t', and allocating more and more space for it). □

Corollary 3.3. *For all functions $t(n)$ and $s(n)$, such that $s(n) \geq \log t(n)$:*
weak–$\mathrm{CA}_3\mathrm{TIME}\text{-}\mathrm{SPACE}(t(n), s(n)) = \mathsf{weak}$–$\mathrm{CA}_2\mathrm{TIME}\text{-}\mathrm{SPACE}(t(n), s(n))$.

References

[1] A.K. Chandra, D.C. Kozen and L.J. Stockmeyer, Alternation. *J. ACM* **28** (1981) 114–33.
[2] V. Geffert, A communication hierarchy of parallel computations, Elsevier Science. *Theoret. Comput. Sci.* **198** (1998) 99–130.
[3] J. Hromkovič, J. Karhumäki, B. Rovan and A. Slobodová, On the power of synchronization in parallel computations. *Discrete Appl. Math.* **32** (1991) 155–82.
[4] A. Slobodová, Communication for alternating machines. *Acta Inform.* **29** (1992) 425–41.
[5] A. Slobodová, Some properties of space-bounded synchronized alternating Turing machines with universal states only. *Theoret. Comput. Sci.* **96** (1992) 411–19.
[6] P. van Emde Boas, Machine models and simulations, in *Handbook of Theoretical Computer Science*, edited by J. van Leeuwen. Elsevier Science (1989).
[7] J. Wiedermann, On the power of synchronization. *J. Inf. Process. Cybern. (EIK)* **25** (1989) 499–506.

Communicated by J. Hromkovic.
Received June, 2001. Accepted August, 2003.