On the Power of Broadcasting in Mobile Computing

Wiedermann, Jiří
2005

Institute of Computer Science
Academy of Sciences of the Czech Republic

On the Power of Broadcasting in Mobile Computing

Jiří Wiedermann and Dana Pardubská

Technical report No. 944

October 2005

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: (+4202) 6605 3520, fax: (+4202) 86 585 789,
e-mail:jiri.wiedermann@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

On the Power of Broadcasting in Mobile Computing[1]

Jiří Wiedermann[2] and Dana Pardubská[3]

Technical report No. 944

October 2005

Abstract:

A computational model reflecting fundamental computational aspects of wirelessly communicating mobile processors is presented. In essence, our model is a deterministic Turing machine which is able to launch new processes among which a wireless communication via explicitly assigned channels must be programmed. We show that computations of such machines are polynomially time– and space–equivalent to the synchronized alternating Turing machines studied previously in the literature. This shows that nondeterminism can be completely eliminated from synchronized alternation at the price of introducing a program–driven communication among the respective processors.

Keywords:
wireless computing, Turing machines, alternation, complexity

[1]The research was carried out within the institutional research plan AV0Z10300504 and partially supported by grant No. 1ET100300517 and by grant APVT-20-018902

[2]Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic

[3]Department of Computer Science, Comenius University, Mlynská dolina, 842 48 Bratislava, Slovakia, email: pardubska@fmph.uniba.sk

1 Introduction

The recent progress in wireless and mobile information technologies has caused an increased interest in algorithmic aspects of the underlying computing and communicating mechanisms. It seems that so far the respective research has mainly concentrated on the concrete algorithmic issues neglecting almost completely the computational complexity aspects in that kind of computing. This might be due to the lack of formal computational models underlying the wireless mobile computing. If one tries to identify the basic computational and communication properties of the respective mode of computing, after a considerable simplification one arrives at a notion of dynamically reconfigurable nets of mobile processors which can communicate one with each other via a "radio". On a sufficiently high level of abstraction this can be modelled as though the processes communicated over a set of channels, with messages broadcasted over different channels being "heard" by any other processes tuned to the respective channels.

What can be said about the computational power and efficiency of the "computational model" we have just sketched? Does wireless mobile computing as captured by the simplified model bring new quality into computing when compared to the classical ways of computing? Answers to these questions will present the main motivation for this paper. In order to get such answers we will devise a formal model capturing the main features of mobile wireless computing and we will investigate its computational efficiency. Our model is a parallel deterministic Turing machine that is able to spawn parallel processes which can create communicating groups. The main design idea of our model has been its transparency as far as the communication mechanism among the processes is concerned. That is, in our model establishing a connection within different subsets of processors requires explicit allocation of different communication channels to the respective processes. Moreover, the same mechanism of inter–processor communication is also used for detecting termination condition. The above mentioned transparency makes visible (and thus, chargeable) the activities which e.g. in the case of alternating Turing machines occur "behind the scene" and bring free benefits.

In our investigations we have been primarily inspired by the concept of alternation (cf. [1]). In the complexity theory this concept is seen as a theoretically neatest framework for studying parallel computations. It might look as quite far–fetched to try to see the alternating processes as wirelessly communicating mobile processes, but it is not entirely so. In a classical alternating Turing machine, the "wireless" communication among its running processes is used in order to determine the termination of the whole computation. The mobility of processes is captured by their dynamic emergence and extinction and by the fact that they are not associated with any concrete location.

The second source of inspiration for our work has been offered by so–called synchronized alternation which is a more general concept than the classical alternation. Synchronized alternating Turing machines have been investigated since the end of nineteen nineties ([5], [3]). The motivation behind these studies was similar to ours: what is the benefit of an additional communication among processes of a running alternating machine. In our terms, in the original works on synchronized alternation (cf. [5] and [3]) but a single channel communication was considered. It appeared that the respective machines had the same time efficiency as the classical alternating Turing machines, but they were more space–efficient than the latter machines: their logarithmic space had the same power as their polynomial time. This is the property not known to hold for the classical alternating Turing machines.

In our modelling we have disposed both of nondeterminism and of the acceptance mechanism which both are crucial ingredients of alternation (either classical or synchronized one). Instead, we introduced a versatile deterministic inter–processor message exchange mechanism. Surprisingly, our main result states that these changes compensate for the loss of the respective abilities. In fact, our entirely deterministic parallel machines are equivalent to the synchronized alternating Turing machines.

We are far from claiming that our model represents an ideal model of mobile wireless computing. Nevertheless, we believe that the benefit of having such a model is threefold. First, our model characterizes the computational power of a certain type of wireless mobile computing. Second and perhaps more importantly, the model shows that nondeterminism and the acceptance mechanism of classical alternating machines are not necessary in order to get a full computational power of synchronized Turing machines. Last but not least, the new model leads to an alternative "machine dependent"

characterization of PSPACE and EXPSPACE, or APTIME and AEXPTIME, respectively. Namely, it shows that synchronized alternation is equivalent to deterministic parallelism enhanced by an explicit communication mechanism.

The paper is structured as follows. In Section 2 we present our model of the so–called wireless parallel Turing machine and introduce the respective complexity measures. Next, in Section 3 we state the main complexity characterizations of computations by such a machine. Finally, in Section 4 we review the achievements of the paper.

2 Wireless Parallel Turing Machine

In order to arrive at a computational model in which processes communicate via broadcasting, we use the multiplicative ability of processes of an alternating Turing machine in universal states and enhance the latter machine by a special mechanism which enables "wireless" information transfer among the processes which have "tuned in" the same channel. We will also dispose of the acceptance mechanism of alternating machines whose activity will be substituted by a versatile communication mechanism.

Definition 2.1 *A k–tape wireless parallel Turing machine (WPTM) with a separate read–only input tape and a separate channel tape is an eleven–tuple $M = (k, Q, R, \Sigma, \Gamma, \Delta, q_0, r_0, \varepsilon, q_{accept}, q_{reject})$ where*

- *k is the number of work tapes;*
- *$Q \times R$ is the finite set of states;*
  - *Q is the set of working states with* initial state $q_0 \in Q$;
  - *R is the set of communication states also containing four distinguished states: initial communication state r_0, empty communication state ε and states q_{accept}, q_{reject} which are* accepting *and* rejecting *states, respectively;*
- *Σ is a finite input alphabet ($\$ \notin \Sigma$ is an endmarker);*
- *Γ is a finite work tape alphabet ($\sharp \in \Gamma$ is the blank symbol, $\sharp \notin \Sigma$);*
- *$\Delta \subseteq Q \times R \times (\Sigma \cup \{\$\}) \times \Gamma^{k+1} \times Q \times R \times (\Gamma - \{\sharp\})^{k+1} \times \{-1, 1\}^{k+2}$ is the next move relation.*

The elements of Δ are called transitions. The machine has a read–only input tape with endmarkers, k work tapes and one channel tape. The work tapes and the channel tape, jointly be referred to as tapes, are initially blank. The tapes are unbounded to the right, with their cells numbered from 0.

Let $\delta = \langle q, r, x, a_1, \ldots, a_{k+1}, q', r', a'_1, \ldots, a'_{k+1}, d_1, \ldots, d_{k+2} \rangle \in \Delta$ be a transition of M. According to this transition in a single *step* M finding itself in working state q, in communication state r, reading symbol x from the input tape and a_i from the i-th tape, for $i = 1, 2, \ldots, k+1$, enters a new working state q', new communication state r', writes symbol a'_i on the i-th tape and moves each of the $k+2$ heads in direction d_j (left or right) one tape cell, for $j = 1, 2, \ldots, k+2$.

A *configuration* of a WPTM M is an element of $Q \times R \times \Sigma^* \times ((\Gamma - \{\sharp\})^*)^{k+1} \times \mathbb{N}^{k+2}$, representing the working and communication state of the finite control, the input, the non–blank contents of $k+1$ tapes, and $k+2$ head positions.

A *head configuration* of M is an element of $Q \times R \times (\Sigma \cup \{\$\}) \times \Gamma^{k+1}$ representing the working and communication state of the finite control and the contents of cells scanned by each head.

We say that a transition with the new communication state $r' \neq \varepsilon$ broadcasts state $r' \in R$. Such a transition is called a *broadcasting transition*. There is one syntactic restriction holding for broadcasting transitions called "unanimous broadcast rule": *the broadcasting transitions pertinent to the same head configuration must all broadcast the same communication state.* They can differ in the remaining parts, i.e., they can prescribe entering different working states and different rewritings and movements. We say that a configuration is *tuned to channel* c if it has string c written to the left from the current channel tape head position. If c is a non–empty string, then it is also called a *channel number*. A configuration tuned to c to which a transition with the new communication state $r' \neq \varepsilon$

applies is said to broadcast r' on channel c. A configuration broadcasting ε is effectively considered as no–broadcasting (or *silent*) configuration (cf. the definition of a transition modified by a broadcasting below). The silent transitions are useful in situations when a process "does not want" to broadcast any information, e.g., when re–tuning its channel.

A configuration β is a δ–successor of a configuration α with respect to transition $\delta \in \Delta$ (written as $\alpha \vdash^{\delta} \beta$) if β follows from α in one step, according to transition δ. The move $\alpha \vdash^{\delta} \beta$ is called a *simple step* of M.

A configuration without successors is called a *terminal configuration*. Note that a configuration can have several different δ–successors.

In order to define a computation of M we need a couple of further preliminary definitions.

Function $Tuned : Q \times R \times \Sigma^* \times ((\Gamma - \{\sharp\})^*)^{k+1} \times \mathbb{N}^{k+2} \to (\Gamma - \{\sharp\})^*$ assigns to each configuration its channel number.

In a similar vein we define function $Broadcast : Q \times R \times \Sigma^* \times ((\Gamma - \{\sharp\})^*)^{k+1} \times \mathbb{N}^{k+2} \to R$ returning to each configuration the unique state broadcasted by the transition applicable to that configuration (remember the "unanimous broadcast rule"). With a slight abuse of notation this function will naturally be extended to a set $L \neq \emptyset$ of configurations as follows:

$$Broadcast(L) = \begin{cases} b & \text{iff for all } \alpha, \beta \in L, Tuned(\alpha) = Tuned(\beta) \\ & \text{and } Broadcast(\alpha) = b \\ \perp & \text{otherwise} \end{cases}$$

(symbol $\perp$ denotes an undefined value).

Finally, we define projection $Comm : Q \times R \times \Sigma^* \times ((\Gamma - \{\sharp\})^*)^{k+1} \times \mathbb{N}^{k+2} \to R$ assigning to each configuration its communication state.

For any communication states $u, v \in R$ and any configuration α in communication state u the notation $\alpha|_{u:=v}$ denotes configuration α in which state u is changed to v.

Let L be a set of configurations, $L_c \subseteq L$ a subset of configurations tuned to c and $\alpha \vdash^{\delta} \beta$ a simple step. Then configuration γ is a so–called δ_L–*successor* of α w.r.t. transition δ *modified by broadcasting* from L (denoted as $\alpha \vdash^{\delta_L} \gamma$) if γ is defined as follows:

$$\gamma := \begin{cases} \beta|_{Comm(\beta):=b} & \text{iff} \quad L_{Tuned(\beta)} \neq \emptyset \quad \text{and} \quad Broadcast(L_{Tuned(\beta)}) = b \\ \perp & \text{iff} \quad L_{Tuned(\beta)} \neq \emptyset \quad \text{and} \quad Broadcast(L_{Tuned(\beta)}) = \perp \\ \beta & \text{iff} \quad L_{Tuned(\beta)} = \emptyset \quad \text{or} \quad Broadcast(L_{Tuned(\beta)}) = \varepsilon \end{cases}$$

Note that the previous three items correspond to the following situations, respectively: in each item we consider broadcasting by configurations in L on a channel to which β is tuned. Case 1 corresponds to the situation when all configurations broadcast unanimously state b. Case 2 deals with the situation when there is a broadcasting conflict on that channel, and case 3 covers the situation when there is no broadcast on that channel.

For any configuration ϱ the *computational graph* $T(\varrho)$ w.r.t. the transition relation Δ of M is a rooted, directed, possibly infinite acyclic multigraph whose nodes are configurations of M and edges correspond to transition and communication links. This graph is defined recursively:

1. ϱ is the root of $T(\varrho)$ at depth $d = 0$;

2. Let C_d be the set of configurations at depth $d \geq 0$. Then for all non–terminal configurations $\alpha \in C_d$, set C_{d+1} contains all δ_{C_d}–successors of α; i.e., set $C_{d+1} = \{ \gamma \mid \exists$ non–terminal $\alpha \in C_d : \alpha \vdash^{\delta_{C_d}} \gamma\}$. If some of the δ_{C_d}–successors of α is undefined, then the whole graph $T(\varrho)$ is undefined.

3. In $T(\varrho)$ there are two kinds of edges:

   - so–called *transition edges* leading from each $\alpha \in C_d$ to each of its δ_{C_d}–successors $\gamma \in C_{d+1}$;
   - so–called *broadcasting edges* leading from each broadcasting configuration $\alpha \in C_d$ to each configuration $\beta \in C_{d+1}$, with $Tuned(\alpha) = Tuned(\beta)$.

Note that the first ("completeness") condition in item 3 as above ensures that $T(\varrho)$ contains, for any configuration α in $T(\varrho)$, the set of all its successors with their communications states possibly modified by broadcasting.

¿From the previous description it is seen that the computational graph is built in a completely deterministic manner: in this graph, all transition and broadcasting edges are defined uniquely. Formally, it is a multigraph since there can be one transition and one broadcasting edge between some nodes of this graph.

A *computational path* starting in configuration ϱ is a (possibly infinite) sequence of configurations of M which are encountered during a traversal down any path in $T(\varrho)$. Any computational path represents a computation of M along that path, or a *process* corresponding to that path.

The nodes of $T(\varrho)$ without successors are called *leaves* of $T(\varrho)$.

A computational graph $T(\varrho)$ of M is a *computational graph accepting input* w if it satisfies the following conditions:

1. *Finiteness:* $T(\varrho)$ is a finite graph;
2. *Initial condition:* $\varrho = (q_0, r_0, w, \underbrace{\nu, \ldots, \nu}_{k+1}, \underbrace{0, \ldots, 0}_{k+2})$ is the *initial configuration* where ν is the null string.
3. *Acceptance agreement:* all leaves of $T(\varrho)$ are terminal configurations at the same depth, in communication state q_{accept} and tuned to the same channel.

In a similar way, the notion of a computational graph rejecting the input string is defined.

¿From practical point of view the acceptance agreement means that all processes share the information that all of them have accepted the input. If the termination time is unknown then the acceptance in the same depth can be achieved e.g. via barrier synchronization. The idea is to check periodically and synchronously in all processes, their "readiness" to terminate the computation (cf. [4]).

We say that M *accepts* w if M's computational graph accepts w; we define $L(M)$ to be the set of strings accepted by M.

The working space (channel space) complexity of a configuration is the sum of length of the non–blank contents of corresponding work tapes (the length of the channel tape). The working space of a computational graph T is the maximum work space of any configuration in T; the channel space of T is defined similarly. The time of T is the maximum length of any path in T.

A WPTM M *operates in work space* $S(n)$ (*channel space* $C(n)$) if for every string $w \in L(M)$ of length n there is a computational graph of M of working space at most $S(n)$ (channel space $C(n)$) that accepts w. Similarly, M *operates in time* $T(n)$ if for every string $w \in L(M)$ of length n there is a computational graph of M of time at most $T(n)$ that accepts w.

The introduction of a separate space measure based on the size of the channel tape is motivated by the wireless mobil computing. The channel space measure hints to the size of the respective communication mechanism.

The resulting model can also be characterized as a co–nondeterministic machine enhanced by a certain processor inter–communication mechanism and without the acceptance mechanism of alternating Turing machines: the acceptance criterion must be programmed in the machine's instructions. The deterministic nature of a WPTM contributes to the "realness" of this model as far as the wireless mobil computations are concerned.

When the transition relation becomes a function it is easily seen that a WPTM turns into the classical deterministic Turing machine. Similarly, it is an easy exercise to realize that a WPTM can simulate in linear time a nondeterministic or a co-nondeterministic Turing machine. Although for both machines the acceptance criteria are different, the WPTM can easily accommodate any of them. In fact, we will show that the WPTMs and alternating Turing machines are polynomially time related.

3 The power of the wireless communication

We start by comparing the WPTMs to the classical ATMs.

Theorem 3.1 *Let $T(n)$ be a time constructible function with $T(n) \geq n$. Let A be an alternating Turing machine of time complexity $T(n)$. Then there is a WPTM M simulating A in time $O(T(n))$ and in working and channel space $O(T(n))$.*

Sketch of the proof: W.l.o.g. assume that in the computation tree T_A of A all branchings are of order at most 2. Design M as follows: starting from the initial configuration of A, M applies to it all applicable transitions of A irrespectively whether the configuration was an existential or a universal one, and in this manner proceeds as though descending the levels in T_A. However, in their working memory the processes in M also remember the so–called *routing information* representing the computational path in T_A from its root to a configuration c at depth d describing the respective process at time d. The computational path to c is represented as a binary number of form $(b_1, \ldots, b_d)_2$, with $b_i \in \{0, 1\}$. This number is also called the *path number* of c.

A part of routing information is the information on the type of T_A's nodes (existential or universal) on that path and a variable *qual* in which the so–called *quality* of each node along the path is stored. The quality of a node in T_A is an element from the set $\{\perp, Y, N\}$ denoting undefined, accepting and rejecting value. For a node v, the quality of v refers to the computational subtree T_v of T_A rooted at v. If T_v is an accepting (rejecting) tree, then the quality of v is defined as Y (N); otherwise, the quality gets undefined, denoted as $\perp$ (cf. [1]). Thus, the routing information for a configuration c stored at v is of form $\langle (b_1, \Box_1, qual_1), \ldots, (b_d, \Box_d, qual_d) \rangle$, with $b_i \in \{0, 1\}$, $\Box_i \in \{\exists, \forall, |, \}$ and $qual_i \in \{\perp, Y, N\}$. The quantifier $\exists$ denotes the existential branching at v, $\forall$ the universal branching (with two successors) and $|$ denotes "no branching" — a deterministic configuration having but one successor. Routing information description is of length $O(T(n))$ at most.

After simulating the $T(n)$ steps (remember that $T(n)$ has been a time constructible function) of A, the processes of M start to verify the acceptance condition for A. Essentially, this condition states how the "answers" from the individual terminated processes must be combined while "climbing" towards the root of T_A (cf. [1]). The "climbing" is based on the routing information. In order to proceed from a given level d to the next upper level $d - 1$, we must first have "filled in" the values of $qual_d$ in all processes in M.

In the sequel we will use the following notation: $S_{\langle b_1, \ldots, b_d \rangle}$ denotes the set of the processes of M whose path numbers share the prefix $(b_1, \ldots, b_d)_2$. Note that in the routing information of processes in $S_{\langle b_1, \ldots, b_d \rangle}$, the values of $\Box_i$ for $i = 1, \ldots, d$, coincide.

The values of $qual_d$ are computed in rounds. In a so–called *d-round,* for $d = T(n), T(n) - 1, \ldots, 1$, the value of $qual_d$ is computed for each process in M. We show that at the end of d-round the following invariant holds: for each process in $S_{\langle b_1, \ldots, b_d \rangle}$, the values of $qual_i$, for $i = T(n), T(n) - 1, \ldots, d$ have already been determined and, moreover, in all processes in $S_{\langle b_1, \ldots, b_d \rangle}$ the value of $qual_d$ is the same.

The algorithm starts with $d = T(n)$ by assigning values Y, N, or $\perp$ to $qual_{T(n)}$ in each process of M which has accepted, rejected, or has not finished its task, respectively. Since for all path numbers $S_{\langle b_1, \ldots, b_{T(n)} \rangle}$ are the singleton sets, the invariant is restored at the end of the $T(n)$-round.

Assume now that the invariant holds at the end of a d-round, $T(n) \geq d > 1$. We show the algorithm restoring the invariant at the end of a $(d - 1)$-round.

Each process in $S_{\langle b_1, \ldots, b_d \rangle}$ inspects the value of $\Box_d$ in its routing information.

If it was "$|$," then the process sets $qual_d := qual_{d+1}$.

If it was "$\forall$" or "$\exists$," then there exist two "brother" sets of processes $S_{\langle b_1, \ldots, b_d, 0 \rangle} = S_1$ and $S_{\langle b_1, \ldots, b_d, 1 \rangle} = S_2$ within $S_{\langle b_1, \ldots, b_d \rangle} = S$. Clearly, S_1 and S_2 are disjoint, $S_1 \cup S_2 = S$ and for all processes in S_1, $qual_{d+1}$ has the same value, and the same holds for S_2. In order to compute $qual_d$ in S from knowing $qual_{d+1}$ both in S_1 and S_2, respectively; both sets must broadcast the values of their respective $qual_{d+1}$ to all processes in S. The broadcasting is performed over channel no. $(b_1 \ldots b_d)_2$: first, the processes from S_1 broadcast their value of $qual_{d+1}$ to processes of S and the same is then done by processes of S_2. Due to the fact that the values of $qual_{d+1}$ are the same within each S_1 and S_2, respectively, no broadcasting conflict can happen (as required by condition 2(a) in the definition of the complete computational tree of M in Section 2). As a result, in accordance with the rules for quality assignment in alternation trees (cf. [1]) each process in S can compute its quality at depth d. Obviously, all processes in S will get the same value for $qual_d$. Note that for all sets $S_{\langle b_1, \ldots, b_d \rangle}$ all the necessary communication is carried out over the disjoint channels. The round is finished by setting $d := d - 1$.

In a similar way we proceed level by level, until we get $d = 1$ and in all processes, $qual_1$ gets its final value. Based on this, a broadcasting transition on channel 0 into the accepting communication state q_{accept} or q_{reject} can be realized in all processes and the simulation can terminate.

We conclude that in $O(T(n))$ steps the acceptance condition for A can be verified by M, indeed. □

Theorem 3.2 *Let $T(n)$ be a time constructible function with $T(n) \geq n$. Let M be a WPTM of time complexity $T(n)$. Then M can be simulated by an alternating Turing machine A of time complexity $O(T^2(n))$ and space complexity $O(T(n))$.*

Sketch of the proof: Let T_M be the computational graph of M on input w. T_M is a superposition of two graphs: w.l.o.g. we can assume that the first is the binary tree — called the *spawning tree* hereafter — underlying T_M and capturing the processes spawning history. The second graph is the *broadcasting graph* capturing the broadcasting of communication states from the broadcasting configurations to the respective target configurations. If T_M is an accepting computational graph, then, w.l.o.g., the leaf configurations are tuned to channel 0.

The idea of the simulation algorithm is to guess accepting configurations and to check that these configurations are leaves of an accepting computational graph T_M for input word w. Note that the leaf configurations are uniquely determined by M and w thanks to the fact that any WPTM is a fully deterministic device.

To verify the before mentioned property we will first guess the accepting configurations c_v in the leaves of T_M by creating pairs (path number p_v, accepting configuration c_v). Then, we will reconstruct T_M in a bottom-up manner by checking in parallel for each leaf v in T_M that the computational path with path number p_v ends up in configuration c_v. In fact, we will traverse the computational path in a bottom-up manner, from the leaf to the root and verify that the path ends up in the initial configuration. Within this process we will have to repeatedly guess and verify the subgraphs of T_M consisting of all paths leading from a given node towards the root. Due to the deterministic nature of M's computation, such paths are unique and therefore their repeated traversal will be possible.

To better understand how the bottom–up traversal is done let us focus on one computational step of M first. Let v be a vertex of T_M at depth d, let p_v, c_v be the corresponding path number and configuration, respectively. Note that $d = |p_v|$, with $|p_v|$ denoting the length of the (binary) representation of p_v. With the exception of the communication state, configuration c_v is uniquely determined by the configuration in the predecessor v_1 of v in the spawning tree. The communication state of v is determined by the set $S_{(|p_v|-1,Tuned(c_v))}$ of v's predecessor vertices in the broadcasting graph (see Fig. 1). For a correct realization of a computational step the condition of unanimous broadcasting must be satisfied, i.e., all vertices from $S_{(|p_v|-1,Tuned(c_v))}$ must broadcast the same communication state $Comm(c_v)$.

So, in order to proceed from v to v_1 we have to (i) *guess* a configuration c_{v_1}, (ii) *guess* a vertex $v_2 \in S_{(|p_v|-1,Tuned(c_v))}$ and (iii) verify that all vertices from $S_{(|p_v|-1,Tuned(c_v))}$ broadcast the same communicating state $Comm(c_v)$.

The simulation algorithms stars in a so-called *preparatory phase* in which A starts its work by first setting $cons \leftarrow 0$ for the acceptance agreement and then, in universal mode, A creates $2^{T(n)}$ processes at depth $T(n)$. It is here where we need the constructibility of $T(n)$. For each process A computes its path number and guesses the respective terminal leaf configuration in T_M with the channel number $cons$ and communication state q_{accept}. Thanks to the fact that all the configurations are accepting ones no broadcasting conflict can occur at depth $T(n)$. Then, in every process just created procedure $Reachability(p_v, c_v)$ is issued.

Procedure $Reachability(p_v, c_v)$ is quite a complex recursive procedure which in turn calls three other procedures. Its goal is to verify recursively whether a configuration c_v is reachable from the root via a specified computational path p_v assuming that no broadcasting conflicts occur in T_M whenever necessary.

$Reachability(p_v, c_v)$ works as follows.

If $|p_v| = 0$ then there cannot be any broadcasting from configurations at a higher level and hence $Reachability(\lambda, c_v)$, with λ denoting the empty string of length 0, straightforwardly verifies whether c_v equals the initial configuration or not and returns the corresponding answer.

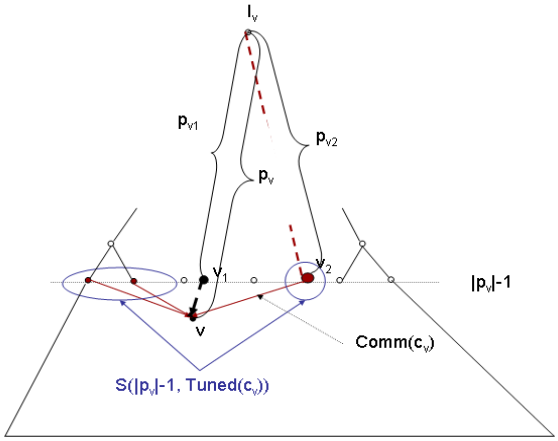

Figure 3.1: The schema of broadcasting in T_M

If $|p_v| > 0$, from p_v the process computes $p_{v_1} = p_v \div 2$, guesses c_{v_1} and then existentially splits into two branches.

The first branch corresponds to the case when $c_{v_1} \vdash^{\delta} c_v$, i.e., when the transition δ from v_1 to v was a simple move (not using any broadcasting). Thus we have to verify the existence of $\delta \in \Delta$ such that $c_{v_1} \vdash^{\delta} c_v$, that c_{v_1} is reachable (via path p_{v_1}) and that there was no broadcasting on channel $Tuned(c_v)$. The first condition is checked by trying all δ's for $c_{v_1} \vdash^{\delta} c_v$. Once a matching δ is found we verify the remaining two conditions by splitting universally and calling $Reachability(p_{v_1}, c_{v_1})$ on one branch and procedure $UndisturbedBroadcast(|p_{v_1}|, Tuned(c_v), \varepsilon)$ on the other branch (see the description of procedure $UndisturbedBroadcast$ in the sequel). If the move $c_{v_1} \vdash^{\delta} c_v$ could not been verified for any δ, then the procedure rejects.

The second branch corresponds to the case when state $Broadcast(c_{v_2}) = b$ (say) was broadcasted to c_v on channel $h = Tuned(c_v)$. In this case $Reachability$ guesses p_{v_2} and c_{v_2} such that $Tuned(c_{v_2}) = h$. Then it goes deterministically through all $\delta \in \Delta$ in order to find δ such that $v_1 \vdash^{\delta_{\{v_2\}}} v$. If such δ is found, the procedure splits universally into three processes performing in parallel procedure $Reachability(p_{v_1}, c_{v_1})$, procedure $Reachability(p_{v_2}, c_{v_2})$, and procedure $UndisturbedBroadcast(|p_{v_2}|, h, b)$, respectively. Otherwise, procedure $Reachability$ rejects.

$UndisturbedBbroadcast(d, h, b)$ verifies that no other state than b is broadcasted on channel h at depth d in T_M. The other way round, no configuration tuned to h and broadcasting $b' \neq b$ is reachable in T_M at depth d.

To check that the latter property holds assume $B_{(d,h,b)}$ is the set of all configurations which could in principle occur in T_M at depth d tuned to h broadcasting a communication symbol b (recall that the configuration is said to broadcast a symbol b if a transition broadcasting b applies to that configuration). We will in parallel check for each element $c \in \bigcup_{b' \neq b} B_{(d,h,b')}$, i.e., for each path p, with $|p| = d$, and each $b' \neq b$ that there is no way for path p starting in the initial configuration to end up in c while broadcasting b'.

In order to do so the procedure splits universally, systematically creating a process for each path number p between $(\underbrace{0, \ldots, 0}_{d})_2$ and $(\underbrace{1, \ldots, 1}_{d})_2$, each communicating state $b' \in R$, $b' \neq b$ (or $b' \in R$ iff $b = \varepsilon$, respectively), and configuration c of size at most d such that c broadcasts b' on channel h. On each such a branch $Nonreachability(p, c)$ (described in the sequel) is called in order to verify that in T_M configuration c is not reachable via path p.

$Nonreachability(p, c)$ checks that the statement *"in T_M, path p starting in initial configuration leads to c"* is false. In fact, we want $Nonreachability(p, c) = \neg(Reachability(p, c))$ to hold. Hence, $Nonreachability(p, c)$ works much like $Reachability(p, c)$ with the substantial difference that existential states are replaced by universal ones and vice versa and instead of $UndisturbedBroadcast$

procedure *Broadcast* (described in the sequel) is called. $Nonreachability(p, c)$ accepts if and only if $Reachability(p, c)$ called with the same parameters rejects or is not defined.

$Broadcast(d, h, b)$ verifies that state b is broadcasted on channel h by some configuration at depth d in T_M. The other way round, in T_M at depth d there exists a reachable configuration tuned to h and broadcasting b.

To check the property for each configuration $c \in S_{d,h,b}$ and path number $p \in \{0,1\}^d$ a process is existentially created. Then, $Reachability(p, c)$ is called in each such process.

The simulation of M by A ends successfully by accepting the input if and only if all calls of checking procedures issued in the verification phase end successfully.

All four above mentioned procedures are described in the Appendix in greater detail.

Now a few words regarding the correctness of the simulation algorithm are in order. The preparatory phase obviously serves its purpose. Subsequently, procedure *Reachability* was used to ensure the reachability of the nodes of T_M from the root. However, this property has been checked under the condition that there were no broadcasting conflicts in T_M. Such conflicts, if any, were detected by $UndisturbedBraodcast$ by inspecting all configurations in T_M possibly "jamming" a state broadcasted on a given channel at the given level. In this way the correctness of guesses of *Reachbility* concerning configurations in T_M modified by broadcasting on the given channel has been either verified or a "mismatch" in the broadcasted communication states has been detected, if any.

The formal proof of correctness can be done by mathematical induction showing that

- $Reachability(p_v, c_v)$ returns **true** iff vertex v with path number p_v corresponds to configuration c_v in computational tree $T_M(c_I)$ with initial configuration c_I in its root;
- $UndisturbedBroadcast(d, h, b)$ returns **true** iff no other state than b is broadcasted on channel h at depth d in $T_M(c_I)$;
- $Nonreachability(p_v, c_v)$ returns **true** iff the configuration in vertex v with path number p_v is different from c_v, in $T_M(c_I)$;
- $Broadcast(d, h, b)$ returns **true** iff state b is broadcasted on channel h at depth d in $T_M(c_I)$.

Finally we determine the complexity of A. Obviously, the preparatory phase is of time complexity $O(T(n))$. The complexity of the next two phases is quadratic w.r.t. $T(n)$ since by each call of the checking procedures the depth parameter in the calls decreases by 1 until the calls terminate at depth 0. However, preparing the necessary parameters for the calls also takes time $O(T(n))$ which leads to the quadratic time complexity of the verification phase.

□

In what follows, in addition to the standard deterministic complexity classes such as $\mathsf{LOGSPACE}$, PTIME, PSPACE, and $\mathsf{EXPTIME}$ we will also use their analogues defined for the alternating and wireless parallel Turing machines. These classes will be denoted by prefixing the standard complexity classes listed before by A or W, respectively.

The following corollary characterizing the power of WPTM's polynomial time is the consequence of both previous theorems and of the known properties of alternating complexity classes (cf. [1]).

Corollary 3.1 *For any time constructible function $T(n)$ with $T(n) \geq n$,*

$$\bigcup_{k>0} \mathsf{WTIME}(\mathsf{T}^k(\mathsf{n})) = \bigcup_{k>0} \mathsf{ATIME}(\mathsf{T}^k(\mathsf{n}))$$

i.e.,

$$\mathsf{WPTIME} = \mathsf{APTIME} = \mathsf{PSPACE}$$

Next, we turn our attention to the relation between the deterministic and wireless mobile space.

Theorem 3.3 *Let D be a deterministic Turing machine of space complexity $S(n)$. Then D can be simulated by a WPTM M of channel space complexity $O(\log S(n))$ and of working tape complexity $O(1)$.*

Sketch of the proof: The proof mirrors the proofs of similar theorems known in the theory of synchronized alternation (cf. [2], [5]). I.e., the contents of the cells of a single tape of D are kept in processes of M. A new proces is spawned each time when the head of D on its work tape enters a blank cell. The cells are numbered by their position number and so are the processes. Thus, on their working tape, process No. i remembers the symbol written in the i-th cell, the presence of the working head of D at this cell (yes or no), and the state of D if the head scans the i–th cell of D. Number i in binary is stored at the channel tape of the i–th process. M simulates D by following its moves and by updating the information in the processes corresponding to the respective updated cells and state changes of D. The "head movements" from the i–th process (cell) are realized by broadcasting the respective "message" to the cell's left or right neighbor on channel $(i-1)$ or $(i+1)$ whose number equals the index of that cell. Thus, re–tuning a channel amounts to adding or subtracting 1 from the current channel number. Clearly, the channel space complexity of the simulating machine is $O(\log S(n))$ while the working space complexity is $O(1)$.

□

Theorem 3.4 *Let $S(n) \geq \log n$. Let M be a WPTM of both working and channel space complexity $O(S(n))$. Then there exists $c > 0$ such that M can be simulated by a deterministic Turing machine D in space $O(c^{S(n)})$.*

Sketch of the proof: Consider the computational graph T_M of M on input w. Each configuration in T_M is of size $O(S(n))$; in this estimate, the input head position is included thanks to our assumption on the size of $S(n)$. If w is accepted by M, then at most $O(c_1^{S(n)})$ different processes can occur in M. It follows that for a suitable $c > c_1$ D has enough space to write down all the respective configurations and within this space it can easily keep track of all M's actions. D will accept if and only if its computation will correspond to an accepting tree of M on w.

□

¿From theorems 3 and 4 and from Corollary 1 we get further characterizations of WPTM complexity classes.

Corollary 3.2 *For any $S(n) \geq \log n$*

$$\mathsf{WSPACE(S(n))} = \bigcup_{\mathsf{c>0}} \mathsf{DSPACE(c^{S(n)})}$$

$\mathsf{WLOGSPACE} = \mathsf{PSPACE} = \mathsf{WPTIME}$, $\mathsf{WPSPACE} = \mathsf{EXPSPACE} = \mathsf{WEXPTIME}$

¿From Corollary 1 and 2 we can see that the fundamental complexity classes, viz. logarithmic space and polynomial time, coincide both for synchronized alternating (cf. [5] and [3]) and wireless parallel Turing machines. Thus, w.r.t. these classes both models are equivalent. Especially note that similarly as synchronized alternating Turing machines, WPTMs use their space in an optimal manner — e.g., "wireless" polynomial time equals "wireless" logarithmic space.

4 Conclusion

The effort of designing a computational model of wireless mobile nets has brought an unexpected result. In addition to the original goal of characterizing the computational power of a certain kind of such nets, our results have also thrown new light on the nature of the classical and synchronized alternation as that of a computational resource. While the (synchronized) alternation machines with their unrestricted use of nondeterminism and their fancy acceptance mechanism look as a quite unrealistic computational model we have shown that these machines are in fact equivalent to deterministic parallel devices possessing the ability of information exchange via broadcasting on unbounded number

of different channels. On the one hand, these results rank the wireless mobile nets under our consideration among the very powerful computational devices. On the other hand, the same results confirm the fact that the very idea of alternation is not so far from the reality as it might appear at the first sight.

In future it would be of interest to study the computational power of nondeterministic wireless parallel Turing machines.

Bibliography

[1] Chandra, A., Kozen, D., Stockmeyer, L.: Alternation. Journal of the ACM, Vol. 28, pp. 114–133, 1981.

[2] Geffert, V.: A Communication Hierarchy of Parallel Computations. Theor. Comput. Sci., Vol 198, No. 1-2, pp. 99–130, 1998

[3] Hromkovič, J., Karhumäki, J., Rovan, B., Slobodová, A.: On the power of synchronization in parallel computations. Discrete Appl. Math. 32 (1991), 155-182

[4] Valiant, L. G.: A Bridging Model for Parallel Computation, Communication of the ACM, Vol. 33 No. 8, pp. 103–111, 1990

[5] Wiedermann, J.: On the Power of Synchronization. Elektronische Informationsverarbeitung und Kybernetik (EIK), Vol. 25, No. 10, pp. 499–506, 1989

Appendix

This appendix gives a formal description of procedures used in the proof of Theorem 2.
The computation of an ATM A starts by calling procedure *Simulation*. In what follows

M is given by its transition relation Δ; we also assume that a Turing machine constructing $T(n)$ is available to A;

$\mathbf{c}_I$ is a global variable denoting the initial configuration;

existentially A_1 **OR** ... **OR** A_k means an existential branching to a *constant* number of processes $A_1, \ldots, A_k$; analogously, a universal branching is described by **universally** A_1 **AND** ... **AND** A_k

guess(x $\in$ S), guess(x such that P(x)) are the analogues to **existentially** A_1 **OR** ... **OR** A_k. These abbreviations are used in situations, when we want to existentially create process for every object x from a given set S with bounded cardinality (the set S can as well be specified by some condition P) . As the cardinality of branchings in ATM is bounded by a constant, to create a non–constant number of processes the subtree with desired number of leaves is created by repeated existential branchings. We assume that the leaf–creating process "knows" the value x.

Similarly, the constructions **for all(x $\in$ P)**, **for all(x such that P(x))** are the analogues to **universally** A_1 **AND** ... **AND** A_k.

Note that the alternating time consumed by **guess**($x \in P$), or **for all**($x \in P$), respectively, is not a constant but it depends on x and the depth of the tree. In fact, the time complexity in this case equals the logarithm of the number of possible values of x;

$\mathbf{B}_{b,d,h}$ denotes the set of all configurations which could in principle occur in T_M at depth d tuned to h broadcasting a communication symbol b.

```
procedure Simulation(w: input word): boolean;
{ returns true iff WPTM M accepts input word w. }

var cons;        channel number
    c, c_I;      configuration
    p;           path number

begin
c_I ← initial configuration;
cons ← 0;
for all (p such that p ∈ {0,1}^{T(|w|)}) do
    for all (c such that c is terminal configuration tuned to cons) do
        return Reachability(p, c)
    end for
end for
end
```

procedure *Reachability*(p_v: path number, c_v: configuration): boolean;

{ returns **true** iff in computational tree T_M rooted in configuration c_I there is vertex v with path number p_v corresponding to a configuration c_v }

```
var p_{v_1};        path number
    c_{V_1};        configuration
    δ;              transition


    begin
    if |p_v| = 0 then return (c_v = c_I)
    else if |p_v| > 0 then
        p_{v_1} ← p_v ÷ 2;
        guess(c_{v_1});
            existentially(1) OR (2):
(1)         if ∃δ ∈ Δ : c_{v_1} ⊢^δ c_v then
            universally(1a) AND (1b):
(1a)        return Reachability(p_{v_1}, c_{v_1});
(1b)        return UndisturbedBroadcast(|p_{v_1}|, Tuned(c_v), ε);
            else if
                return false
            end if ;
(2)         guess(p_{v_2} such that |p_{v_2}| = |p_v| − 1);
                guess(c_{v_2} such that Tuned(c_v) = Tuned(c_{v_2}));
                    if ∃δ ∈ Δ : c_{v_1} ⊢^{δ_{c_{v_2}}} c_v then
                       universally(2a) AND (2b) AND(2c):
(2a)                   return Reachability(p_{v_1}, c_{v_1});
(2b)                   return Reachability(p_{v_2}, c_{v_2});
(2c)                   return UndisturbedBroadcast(|p_{v_2}|, Tuned(c_{v_2}), Broadcast(c_{v_2}))
                    else if
                        return false
                    end if
    end if
    end
```

procedure $UndisturbedBraodcast$(d: depth,h: channel number,
b: communication state) boolean;

{ returns **true** iff in computational tree T_M rooted in configuration c_I no other state than b is broadcasted on channel h at depth d }

```
var p;      path number
    B;      set of communication states
    b';     communication state
    c;      configuration


begin
if b = ε then B := R
else if B := R \ {b}
end if ;
for all p ∈ {0,1}^d do
    for all b' ∈ B do
        for all c ∈ B_{d,h,b'} do:
            return Nonreachability(p, c);
        end for
    end for
end for
end
```

procedure $Braodcast$(d: depth,h: channel number,
b: communication state) boolean;

{ returns **true** iff in computational tree T_M rooted in c_I state b is broadcasted on channel h at depth d. }

```
var p;      path number
    c;      configuration


begin
guess(p ∈ {0,1}^d);
    guess(c ∈ B_{d,h,b});
        return Reachability(p, c);
end
```

procedure *Nonreachability*(p_v: path number, c_v: confuguration): boolean;

{ returns **true** iff in computational tree T_M rooted in c_I the configuration in vertex v with path number p_v is different from c_v }

```
var p_{v_1};        path number
    c_{v_1};        configuration
    δ;              transition
    b;              communication state


    begin
    if |p_v| = 0 then
       return (c_v ≠ c_I)
    else if |p_v| > 0 then
          p_{v_1} ← p_v ÷ 2;
          for all c_{v_1} do
              universally(1) AND (2):
          end for
(1)       if ∃δ ∈ Δ : c_{v_1} ⊢^δ c_v then
             existentially(1a) OR (1b):
(1a)         return Nonreachability(p_{v_1}, c_{v_1});
(1b)         for all (b ∈ R) do:
                 return Broadcast(|p_{v_1}|, Tuned(c_v), b);
             end for
          else if
               return true
          end if ;
(2)       for all p_{v_2}  do:
              for all (c_{v_2} such that Tuned(c_v) = Tuned(c_{v_2})) do:
                  if ∃δ ∈ Δ : c_{v_1} ⊢^{δ_{c_{v_2}}} c_v then
                     existentially(2a) OR(2b) OR (2c):
(2a)                 return Nonreachability(p_{v_1}, c_{v_1});
(2b)                 return Nonreachability(p_{v_2}, c_{v_2});
(2c)                 for all (b ∈ R \ {Broadcast(c_{v_2})}) do:
                         return Broadcast(|p_{v_2}|, Tuned(c_{v_2}), b)
                     end for
                  else if
                       return true
                  end if
              end for
          end for
    end if
    end
```