RAIRO. INFORMATIQUE THÉORIQUE

GEORG GATI

On schemata and *L* systems for parallel algorithms

RAIRO. Informatique théorique, tome 13, n° 2 (1979), p. 155-184

<http://www.numdam.org/item?id=ITA_1979__13_2_155_0>

R.A.I.R.O. Informatique théorique/Theoretical Informatics
(vol. 13, n° 2, 1979, p. 155 à 184)

ON SCHEMATA AND *L* SYSTEMS FOR PARALLEL ALGORITHMS (*) (1)

by Georg GATI (2)

Communiqué par J. BERSTEL

Abstract. — *The parallel program schemata of Karp and Miller are considered for unbounded parallelism and their main decidability result is extended to this case. The complexity of these schemata is investigated and a scheduling algorithm is presented whose sequential time complexity is polynomial. The scheduling algorithm is applied to a schema representing the Strassen-Winograd algorithm for multiplication of* 2×2 *matrices, where it finds a faster computation than the "obvious" one. Finally it is shown in two examples — matrix multiplication and the knapsack problem — that L systems can characterize essential features of the control and data structure of algorithms. This observation gives rise to the concept of parallel program schema over a* 0 *L system. By reduction to the membership problem for* 0 *L systems, questions like "are certain operations executed simultaneously?" and "can certain data conflicts arise?" can be decided for parallel algorithms represented by this class of schemata.*

Résumé. — *On considère les schémas de programmes parallèles de Karp et Miller pour parallélisme non borné et étend leur résultat principal sur la décidabilité en ce cas. La complexité des calculs parallèles de ces schémas est examinée, et on présente un algorithme de « scheduling » de complexité (séquentielle) du temps polynomiale. L'algorithme de « scheduling » est appliqué à un schéma représentant l'algorithme de Strassen-Winograd pour la multiplication de matrices* 2×2 *où il trouve un calcul plus rapide que le calcul « évident ». Enfin on démontre en deux exemples, la multiplication des matrices et le problème de « knapsack », que les L-langages peuvent caractériser les propriétés essentielles de la structure du contrôle et des données des algorithmes. Cette observation donne lieu à un concept formel de schéma d'un programme parallèle sur un* 0 *L-langage. Par réduction au problème du « membership » pour* 0 *L-langages on peut décider des questions telles que « est-ce que certaines opérations sont exécutées simultanément ? » et « certains conflits de données peuvent-ils se produire? » pour les algorithmes parallèles représentés par cette classe de schémas.*

0. INTRODUCTION

The concept of parallelism is a central theme in the theory of algorithms and the theory of programming. This work is an attempt to treat parallelism in a formal apparatus that also permits the application of complexity considerations

(*) Reçu mai 1978 et dans sa version définitive février 1979.
(1) Part of the results of this work has been announced in [7].
(2) Institut für Informatik, Eidgenössische Technische Hochschule Zürich, CH-8092 Zürich, Schweiz.

R.A.I.R.O. Informatique théorique/Theoretical Informatics, 0399-0540/1979/184/$ 4.00

and of results of the theory of "parallel" formal languages, namely of the theory of L systems. Because of the large number of existing formal systems, we do not propose a new one but base our investigations on the parallel program schemata of Karp and Miller [9]. The advances in hardware that have taken place in the meantime make the design of computing machines of very high parallelism more and more promising; therefore it is deemed reasonable to extend the parallel program schemata to unbounded parallelism. From this the question arises how the main decidability result for parallel program schemata concerning the determinacy of an important class of schemata can be extended to the case of unbounded parallelism. This is possible by adding two additional restrictions – local finiteness and compactness – to the restrictions already to be found in [9]. Nevertheless the result still remains a generalization of the one for bounded schemata. Also for the time being it seems that these restrictions will be satisfied in any highly parallel computing machine; another revolution in computer architecture could, of course, obsolete these restricions, too. It should be noted that we have not generalized the other decidability results for parallel program schemata to unbounded parallelism.

As we feel that a general formalism for unbounded parallelism is only satisfactory if it also embraces the viewpoint of complexity, we define a complexity measure for parallel computations ([3]). We present a simple algorithm based on the critical path idea which finds a fastest computation of a schema whose control satisfies certain standard assumptions of scheduling theory, e. g. no cycles are allowed in the precedence graph because this would make scheduling input-dependent. The algorithm is executed on a schema representing Winograd's version [1] of Strassen's algorithm for multiplying 2×2 matrices. Because the control structure of this schema is not trivial, the parallel execution time of a "naively" found parallel computation is longer by the execution time of two parallel additions than the parallel execution time of the parallel computation found by the algorithm. The (sequential) complexity of the algorithm itself can be considered to be cubic. The algorithm is able to operate on an unbounded number of operations.

Due to an idea of Prof. E. Engeler a possible connection between parallel algorithms and "parallel formal languages", namely L systems, is established in two examples, matrix multiplication and the knapsack problem. $D0L$ systems

---

([3]) Remark added after completion of manuscript: In R. A. DE MILLO, K. VAIRAVAN and E. SYCARA-CYRANSKI, *A Study of Schedules as Models of Synchronous Parallel Computation*, J. Assoc. Comput. Mach., Vol. 24, No. 4, 1977, pp. 544-565, a formal model of parallelism is combined with timing considerations from scheduling theory in another way. The model used is related to, but different from the schemata of Karp and Miller.

are explicitly constructed which describe essential features of the parallel algorithms; specifically these are the classes of simultaneously executed operations (which are a main part of the control structure of a parallel algorithm) and the classes of memory locations into which intermediate results of operations are stored simultaneously (which are an essential part of the data and conflict structure of a parallel algorithm).

Finally the above connection between parallel program schemata and L systems is formalized. Results known for the membership problem for certain classes of L systems can be applied to classes of schemata over L systems in order to decide, whether there exists a computation of the schema in which certain operations are executed simultaneously or in which results are stored into certain memory locations simultaneously.

1. SCHEMATA FOR UNBOUNDED PARALLELISM

In this section, we propose to extend the parallel program schemata of Karp and Miller [9] to an unbounded number of (parallel) operations. This would presume a computing facility with unlimited parallel computing resources. The motivation for this assumption is the growing interest in high parallelism. Recent work in this field includes Buchberger [3] and Buchberger and Fegerl [4]. A convincing theoretical model of a highly parallel computing machine is proposed in [3] and the tradeoff in execution performance between sequential execution on a large computer and highly parallel execution on microprocessors is determined for several algorithms of theoretical and practical importance. Aspects of the hardware implementation of the model proposed in [3] are discussed in [4]. Other inspiring research that provides motivation of this work includes Albrecht [2], Schwenkel [14], Sullivan and Bashkow [15] and Sullivan, Bashkow and Klappholz [16].

It is therefore of a certain interest whether the concept of unbounded parallelism can be formulated in a mathematical framework and under which restrictions one can extend results on bounded parallelism to unbounded parallelism.

We choose the parallel program schemata of Karp and Miller because they are very general ([4]) and seem to be well suited for modelling algorithms with unbounded parallelism. The adaptation of parallel program schemata to

([4]) Note added by referee: This was proven in J. L. PETERSON and T. H. BREDT, *A Comparison of Models of Parallel Computation*, in Information Processing, Vol. 74 (I.F.I.P. Congress, Stockholm, 1974), J. ROSENFELD, Ed., pp. 466-470, North-Holland, American Elsevier, Amsterdam, 1974.

unbounded parallelism is simple: in a parallel program schema the number of operations is finite and bounded; in an unbounded schema we permit an unbounded number of operations to be executed by indexing the operations over the natural numbers. Thus in our schemata an unbounded number of different operations can be executed in parallel. We call such a schema an unbounded Karp and Miller schema. We now give the formal definition:

DEFINITION 1.1: An *unbounded Karp and Miller schema* (short: *UKMS*) is a quintuple $S=(\mathrm{Op}, T, D, R, H)$ where $\mathrm{Op}=\{o(i) \mid i \in N\}$ is a set of *operations* indexed over the set of *natural numbers* N. To each operation $o(i)$ we assign an *initiation symbol* $\overline{o(i)}$ and $H(i)$ *termination symbols* $o_1(i), \ldots, o_{H(i)}(i)$ where $H: N \to N_+$ is a map. We write $\underline{o(i)}$ instead of $o_j(i)$, $1 \leqq j \leqq H(i)$, whenever the value of j is irrelevant. Call $F(N) \subset 2^N$ the set of finite subsets of N. Then D, R: $N \to F(N)$ are mappings with the following meanings: It is assumed that the i-th operation $o(i)$ takes its operands from the memory locations whose addresses are in $D(i)$ in the moment of its initiation, which is signalized by the initiation symbol $\overline{o(i)}$, and stores its results into the memory locations of $R(i)$ in the moment of its termination, which is signalized by a termination symbol $o_j(i)$. Also a *decision* with $H(i)$ possible outcomes is made in the moment of its termination; while most programming languages have explicit decision constructs, in UKMS the decisions are merged with the decision-free operations. We call $O := \overline{O} \cup \underline{O}$ the *alphabet* of the schema S where $\overline{O} := \{\overline{o(i)} \mid i \in N\}$ is the set of *initiation symbols* and $\underline{O} := \{\underline{o(i)} \mid i \in N\}$ is the set of *termination symbols*. The *control* $T=(Q, q_0, t)$ consists of a (possibly infinite) set of states Q, an initial state $q_0 \in Q$ and a partial transition function $t: Q \times O \to Q$ which is total on $Q \times \underline{O}$. We also write t for the natural extension of t to $Q \times O^*$.

In a schema the operations are uninterpreted. In order to be able to express concrete computations we have to specify all the data on which the operations can operate, their effects on the data and the decisions. This is done by interpretations; for the definition of this notion we need the following notation: let U be any universe, i. e. the set of all possible contents of the memory locations; a value assignment to the entire memory is then an element of U^N, the countably infinite cartesian product of U with itself; now let $M \subset N$ be a subset of N; then U^M denotes the $|M|$-fold cartesian product of U with itself "the memory locations of M being born in mind". Thus U^M denotes the set of all possible assignments to the memory locations of M.

DEFINITION 1.2: An *interpretation* $I=(U, u_0, F, G)$ of a UKMS S consists of a *universe* U, an *initial contents* of the memory $u_0 \in U^N$ and two families F and G of maps; here $F=(F_i)_{i \in N}$, where $F_i: U^{D(i)} \to U^{R(i)}$ specifies the *change in memory*

contents caused by the i-th operation $o(i)$, and $G=(G_i)_{i\in N}$, where $G_i: U^{D(i)} \to \{1, \ldots, H(i)\}$ specifies the *outcome of* the *decision* associated with $o(i)$; it is clear that F_i must not change the contents of memory locations outside $R(i)$. We write $I(S)$ for the *class of all interpretations* of a schema S.

DEFINITION 1.3: Let us call $Z(I) := Q \times U^N \times ((U^{D(i)})^*)_{i\in N}$ the set of *states* of the UKMS S *under the interpretation* $I=(U, u_0, F, G)$; let $\alpha_0 := (q_0, u_0, (\lambda)_{i\in N})$, where λ denotes the null string, be the *start state* under I. It is then clear how to define a mapping: $Z(I) \times \hat{O} \to Z(I)$ with the help of F, u_0 and t, that is the transition mapping for $Z(I)$. We refrain from giving its specification explicitly here because it is analogous to the one in [9, def. 1.3, 1.4, 1.5].

The next definition describes the elements of the set $C(S, I) \subset O^*$, the *set of computations* of a schema S under the interpretation I. A computation of a schema S is a string of initiation and termination symbols of operations which satisfies certain natural conditions. The main feature of a *parallel* schema is that each set $C(S, I)$ consists of more than one computation, i.e. for a given input there are several (more or less parallel) executions of the program. One possibility is two execute a program strictly sequentially, e.g. $\overline{o(1)}\,\underline{o(1)}\,\overline{o(2)}\,\underline{o(2)}\,\overline{o(3)}\,\underline{o(3)}\,\overline{o(1)}\,\underline{o(1)}, \ldots$, i. e. two or more operations are never initiated simultaneously, but, before the next operation is initiated, all operations having been initiated beforehand must have been terminated. An example for a parallel computation is $\overline{o(1)\,o(2)\,o(3)}\,\underline{o(1)\,o(2)\,o(3)}\,\overline{o(1)}\,\underline{o(1)} \ldots$ where operations $o(1)$, $o(2)$ and $o(3)$ are executed simultaneously.

DEFINITION 1.4: The set $C(S, I)$ of *computations* of the schema S under the interpretation I consists of all strings $x \in \hat{O} := O^* \cup O^\infty$, where O^∞ is the set of infinite strings over O, that satisfy the following conditions (we write $(\alpha \cdot x)$ for "$\alpha \cdot x$ is defined" and $y \leqq x$ for "there is a z in $\hat{O}$ with $yz = x$"):

(1) $(\forall\, y \leqq x)\ (`\alpha_0 \cdot y')$;

(2) $x \in O^* \Rightarrow (\forall\, i \in N)\ (\neg\, `\alpha_0 \cdot x\overline{o(i)}')$;

(3) $(\forall\, s \in O)\ (\forall\, y \leqq x)\ ((\exists\, yz \leqq x)\ (\alpha_0 \cdot yzs) \Rightarrow (\exists\, w \in O^*)\ (yws \leqq x))$.

The conditions below on computations $x \in C(S, I)$ are consequences of definitions 1.3 and 1.4:

(4) $(\forall\, i \in N)\ (\forall\, y \leqq x)\ (\left|\{\overline{o(i)} \in y\}\right| \geqq \left|\{\underline{o(i)} \in y\}\right|)$;

(5) $(\forall\, i \in N)\ (\forall\, r, u, v, w \in O^*).\ (\forall\, t \in \hat{O})$
$((ro(i)\,uo(i)\,vo_j(i)\,wo_k(i)\,t \leqq x \wedge (\forall\, i' \in N)$
$(\underline{o(i')} \in u \Rightarrow R(i') \cap D(i) = \emptyset)) \Rightarrow j = k)$;

(6) Let $y', y'' \in O^*$, and let $y = y' \overline{o(i)}\, y'' o_j(i)$ be a prefix of x, where the indicated occurrence of $\overline{o(i)}$ in y is the initiation symbol corresponding to the indicated occurrence of the termination symbol $o_j(i)$ in y. Denote by $u(y', i) \in U^{D(i)}$ the contents of the memory locations in $D(i)$ after execution of y'. Then $G_i(u(y', i)) = j$.

Let us comment on the meanings of the conditions:

(1) In a computation, only those operations $o(i)$ may be initiated in the state q for which $t(q, \overline{o(i)})$ is defined. This permits one to relate schemata to specific algorithms.

(2) A finite computation must halt in a state in which no operation can be initiated.

(3) This condition is equivalent to the finite delay property [9, def. 1.6 (iii)] for persistent UKMS, *cf.* definition 1.9, however, it is stronger than the finite delay property for general UKMS. It assures the following properties:

(*a*) Between the initiation and the termination of an operation, only a finite number of other operations can be initiated or terminated.

(*b*) Once an operation is ready to be initiated, it will be initiated after the initiation or termination of at most a finite number of other operations.

(4) No operation must be terminated before it is initiated.

(5) If the same operation is initiated twice on the same data, then the outcomes of the decisions associated with the corresponding terminations must be equal.

(6) For different interpretations I and I' of S it might be the case that $x = zo_j(i)\, u \in C(S, I)$ and $x' = zo_{j'}(i)\, u' \in C(S, I')$ while $x \notin C(S, I')$ and $x' \notin C(S, I)$ with $j \neq j'$, $z \in O^*$ and $u, u' \in \hat{O}$. As the state $t(q_0, zo_j(i))$ might be different from the state $t(q_0, zo_{j'}(i))$ the prefix z might be continued differently under different interpretations.

We introduce the sets

$$P(S, I) := \{ y \leqq x \mid x \in C(S, I) \},$$

$$C(S) := \bigcup_{I \in \mathcal{I}(S)} C(S, I) \qquad \text{and} \qquad P(S) := \bigcup_{I \in \mathcal{I}(S)} P(S, I).$$

The sets $P(S, I)$ and $P(S)$ consist of *prefixes* of computations.

If $x \in P(S, I)$ and $|x| \leqq k$ then we write x_k for the k-th symbol in x and ${}_k x$ for $x_1 \ldots x_k$. We write $\Omega(x)$ for the sequence of value assignments $(u_i)_{0 \leqq i \leqq |x|}$ where $u_i := \mathrm{pr}_2(u_0 \cdot x)$, $1 \leqq i \leqq |x|$, and $\mathrm{pr}_2 : Q \times U^N \times ((U^{D(i)})^*)_{i \in N} \to U^N$ is the projection on the second factor. We write $\Omega(x, j), j \in N$, for the subsequence of $\Omega(x)$ which contains only u_0 and those u_i where $x_i = \underline{o(k)}$ with $j \in R(k)$. This notation permits us to formulate the notion precisely that the different parallel computations of a UKMS compute the same results.

DEFINITION 1.5: A UKMS S is *determinate* if

$$(\forall I \in I(S)) \quad (\forall x, y \in C(S, I)) \quad (\forall j \in N) \qquad (\Omega(x, j) = \Omega(y, j)).$$

The above condition for determinacy is a rather strong one because not only the final but also all intermediate results of any two computations $x, y \in C(S, I)$ must be equal and this for all $I \in I(S)$.

DEFINITION 1.6: An interpretation $I \in I(S)$ of a UKMS S is *one-one* if

$$(\forall j, k \in N) \quad (\forall v, w \in U^{(R(k) \cap R(j))}) \qquad (F_j(v) = F_k(w)) \text{ iff } (j = k \text{ and } v = w).$$

We call $I'(S) \subset I(S)$ the class of one-one interpretations of a UKMS S.

LEMMA 1.1: *A UKMS S is determinate iff the condition in definition* 1.5 *is satisfied by all one-one interpretations I of S.*

The proof of this lemma is completely analogous to the proof of [9, lemma 2.2]. The main advantage of the lemma is that the class of interpretations which have to be investigated for equality of the intermediate and final results of the computations is restricted. Now we define a class of UKMS which can be considered as an extension of the parallel flowcharts of [9] to an unbounded number of operations.

DEFINITION 1.7: An r-dimensional *vector addition system* (short: *VAS*) $\mathbf{W}$ is a pair $(d, W) \in N^r \times F(Z^r)$ where $F(Z^r)$ denotes the set of finite subsets of r-dimensional integer vectors. The *reachability set* $R(\mathbf{W})$ of $\mathbf{W}$ is the smallest subset of N^r with:

(1) $d \in R(\mathbf{W})$;

(2) if $v \in R(\mathbf{W})$, $w \in W$ and $v + w \geqq 0$ then $v + w \in R(\mathbf{W})$; where $x = (x_1, \ldots, x_r) \geqq 0$ means $x_i \geqq 0$, $1 \leqq i \leqq r$.

In the following we assume that H is constant on each element N_l of the partition $\mathcal{N}$ of N used in the next definition; thus we can write $H(l)$ instead of $H(i)$, $i \in N_l$. This assumption contains no loss of generality if $(|H(i)|)_{i \in N}$ is bounded; the latter restriction is justified by the observation that at most 2^r outcomes of a decision can be distinguished from the viewpoint of the control.

DEFINITION 1.8: A *UKMS S over* an r-dimensional *VAS* $\mathbf{W} = (d, W)$ with *finitely presented control* (short: *SVAS*) is a UKMS $S = (\mathrm{Op}, T, D, R, H)$ together with the VAS $\mathbf{W}$, a finite partition $\mathcal{N} = \{N_1, \ldots, N_r\}$ of N and two bijective mappings $g_\pm$ satisfying the following conditions:

(1) W consists of exactly two disjoint subsets W_+ and W_- with:

(a) $|W_+| = \sum_{l=1}^{r} H(l)$; $|W_-| = r$;

(*b*) each vector in W_+ has only coordinates from $\{0, +1\}$;

(*c*) each vector in W_- has only coordinates from $\{0, -1\}$;

(*d*) if the i-th coordinate of a vector in W_- is -1 then all other vectors in W_- have as i-th coordinate 0;

(2) g_+ is a bijection from the set $\{(l, j) | l \in \{1, \ldots, r\}, j \in \{1, \ldots, H(l)\}\}$ onto W_+.

g_- is a bijection from $\{1, \ldots, r\}$ onto W_-.

(3) the control T is specified by

$$T := (Q, q_0, t);$$

$$Q := R(\mathbf{W});$$

$$q_0 := d.$$

Then we can specify the transition function t in the following way:

For each $q \in Q$ and $i \in N_l$ we have $'t(q, \overline{o(i)})'$ iff $q + g_-(l) \geqq 0$ and then $t(q, \overline{o(i)}) := q + g_-(l)$; and $t(q, o_j(i)) := q + g_+(l, j)$.

Example 1.1: We specify the control of an SVAS S: $r = 2$, $H(i) = 1$ for all $i \in N$, $\mathcal{N} = \{N_1, N_2\}$, $N_1 = \{1\}$, $N_2 = \{2, 3, 4, \ldots\}$.

$g_+(1,1) = (0,1)$, $g_+(2,1) = (1,0)$; $g_-(1) = (-1, 0)$, $g_-(2) = (0, -1)$. $d = (1,1)$.

Here $C(S, I)$ consists of an infinite number of different elements for all $I \in I(S)$.

As is well known SVAS permit the simulation of the following parallel control structures:

fork: after the termination of an operation a set of operations is initiated simultaneously;

join: after the termination of all of a set of simultaneously executing operations an operation is initiated;

quit: certain operations are terminating but the rest of the simultaneously executing operations continues.

We now explain the background behind this finite presentation of the control. It is rare that one designs an algorithm in which an unbounded number of totally different operations are executed simultaneously. However, an algorithm might produce an "exploding" amount of data items on each of which the same operation has to be performed; e.g. one solves an *NP*-complete scheduling problem for n operations by first producing all permutations of the n operations and then checking, simultaneously for each permutation, whether it is a valid schedule, i.e. respects the precedence graph, etc. In this example, the operations that are initiated in a very big number are all the same from the viewpoint of the control structure of the algorithm; in our case they are even literally the same, but this need not be. It is this type of behaviour that frequently occurs in

algorithms for which unbounded parallelism is desired. One would put all the numbers of the operations that are equal from the viewpoint of the control into the same set N_l of the finite partition of N. The vector $g_-(l)$ corresponds to all initiation symbols $\overline{o(i)}$, $i \in N_l$, and the vector $g_+(l, j)$ to all termination symbols $o_j(i)$, $i \in N_l$. One should bear in mind that the operations corresponding to a member N_l of the partition of N need not be equal or similar with respect to the data or conflicts; in the above mentioned example each permutation of the n operations will be stored in different memory locations. So in an SVAS only the control is finitely presented, but that is not necessary for the conflicts.

We now recall several technical terms from [9].

DEFINITION 1.9: A UKMS is *lossless* if $(\forall i \in N)\ (R(i) \neq \emptyset)$. A UKMS is *repetition-free* if

$$\overline{xo(i)}\ \overline{vo(i)}\ w \in C(S, I) \quad \Rightarrow \quad (\exists \underline{o(j)} \in V) \qquad (R(j) \cap D(i) \neq \emptyset).$$

A UKMS is *persistent* if

$$(\forall q \in Q) \quad (\forall s, s' \in O)$$
$$((s \neq s' \wedge {}'t(q, s)' \wedge {}'t(q, s')') \quad \Rightarrow \quad ({}'t(q, ss')' \wedge {}'t(q, s's)')).$$

A UKMS is *commutative* if

$$(\forall q \in Q) \quad (\forall s, s' \in O)$$
$$(({}'t(q, ss')' \wedge {}'t(q, s's)') \quad \Rightarrow \quad t(q, ss') = t(q, s's)).$$

A UKMS is *permutable* if

$$(\forall q \in Q) \quad (\forall s, s' \in \overline{O}) \quad ({}'t(q, ss')' \quad \Rightarrow \quad {}'t(q, s')').$$

DEFINITION 1.10: We call

$$\mathbf{K}(i, j) := \begin{cases} \emptyset & \text{if } i = j \\ (D(i) \cap R(j)) \cup (D(j) \cap R(i)) \cup (R(i) \cap R(j)) & \text{otherwise} \end{cases}$$

the *domain of conflict* of two operations $o(i)$ and $o(j)$. We define a *conflict relation* $K := \{(i, j) \in N \times N \mid \mathbf{K}(i, j) \neq \emptyset\}$ and a "cross section" $K(i) := \{j \in N \mid \mathbf{K}(i, j) \neq \emptyset\}$. We call $\mathbf{K} := \bigcup_{(i,j) \in N \times N} K(i, j)$ the *set of conflict*

locations of a UKMS S. We call the subset

$$K' := \{(i,j) \in K \mid (\exists I \in I(S))\ (\exists x \in C(S, I))\ (\exists u, v \in O^*)$$
$$(\exists w \in \hat{O})\ (x = u\overline{o(i)}\ v\overline{o(j)}\ w \vee x = u\overline{o(j)}\ v\overline{o(i)}\ w)\}$$

of K the *actual conflict relation and* $\mathbf{K}' := \bigcup_{(i,j)\in K'} \mathbf{K}(i,j)$ the *set* of *actual conflict locations* of a UKMS S and define the "cross section" $K'(i) := \{j \in N \mid (i,j) \in K'\}$.

We remark that an SVAS S is not determinate if

$$(\exists l \in \{1, \ldots, r\})\quad (\exists i, j \in N_l)\quad (i \neq j \wedge (i,j) \in K)$$

and there are computations of S in which operations associated with N_l occur. The first part of this condition is already decidable by inspection of the specification of S, the second part reduces to a decidable problem for vector addition systems [9, cor. 4.1] and we alway assume that the condition is not satisfied.

One immediately sees that the following characterization of determinacy of [9] holds for UKMS, too; we therefore state it without proof:

LEMMA 1.2: *A persistent, commutative, permutable, lossless and repetition-free UKMS is determinate iff*

$$\neg((\exists I \in I(S))\ (\exists x \in P(S, I))\ (\exists (i,j) \in K)\ (x\overline{o(i)} \in P(S, I) \wedge x\overline{o(j)} \in P(S, I))).$$

COROLLARY 1.1: *A lossless and repetition-free SVAS is determinate iff*

$$\neg((\exists I \in I(S))\ (\exists x \in P(S, I))\ (\exists (i,j) \in K)\ (x\overline{o(i)} \in P(S, I) \wedge x\overline{o(j)} \in P(S, I))).$$

Sketch of proof: The proof is analogous to the proof of corollary 2.2 of [9], but one has to observe that SVAS are not persistent "with respect to operations associated with the same element N_l of the partition $\mathcal{N}$ of N". Using definition 1.4, (3), and the assumption below definition 1.10 the "sliding" argument works in this case, too. Details are presented in appendix 1.

We now describe the restrictions on the conflicts that permit us to decide the determinacy of a UKMS.

DEFINITION 1.11: A UKMS is *locally finite* if $(\forall i \in N)(|K'(i)|$ is finite). A UKMS is *compact* if $\mathbf{K}'$ is finite.

Before we discuss the implications of local finiteness and compactness we remark that a locally finite UKMS might not be compact, e.g. consider the mappings $D(0) := R(0) := \{0\}$, $D(i) := R(i) := \{i-1, \quad i, \quad i+1\}$, $i = 1, 2, 3, \ldots$ On the other hand a compact UKMS might not be locally finite, e.g. $D(i) := R(i) := \{0\}$, $i \in \mathbb{N}$.

Definition 1.11 is a contribution to the theory of parallelism by presenting a formal expression for ideas which could be informally described in the following way: One wants to design computing machines in which the number of parallel processes may become arbitrarily large. But then the expenses for enabling and controlling the communication between the single processes grow very fast, too. One therefore tries to restrict this communication in a suitable manner. The restriction caused by local finiteness means that every process can communicate only with a finite number of other processes. As we have seen above one permits an infinite number of conflict locations even when requiring local finiteness: the schema is only *locally* finite. Infinity would then be introduced not only in the number of operations but also in the number of memory locations with potential conflicts. The restriction caused by compactness implies that conflicts can only arise at finitely many memory locations. In the following we also require compactness because we believe that the problems of theoretical investigation and practical administration of infinitely many conflict locations are too difficult for the time being. We also remark that every UKMS which uses only finitely many memory locations is compact.

Let S be a locally finite SVAS such that the partition $\mathcal{N} = \{N_1, \ldots, N_r\}$ of N associated with S has only one element of infinite cardinality, say N_r; several enumerative parallel algorithms fall under this class of schemata. We then require the schema S to be presented in the following way: with each operation $o(i)$ the finite list $K'(i)$ must be specified. We denote by 0_r the r-dimensional zero vector, by δ_{ij} the Kronecker delta and by ${}_rN$ the set $\{1, \ldots, r\}$ of the first r natural numbers. As by the assumption below definition 1.10 $(i, j) \notin K$ if $(i \neq j \wedge \{i, j\} \subset N_r)$ and if operations associated with N_r occur in computations of S, it follows that K' is finite; thus if K' can be effectively constructed and if

$$\rho := \{(l, l') \in {}_rN \times {}_rN \mid (\exists i \in N_l)\,(\exists i' \in N_{l'})\,((i, i') \in K')\},$$

then we can effectively construct ρ. Consider the $2r$-dimensional VAS $\mathbf{W}' = (d', W')$ with $d' := (d, 0_r)$ and

$$W' := \bigcup_{l \in {}_rN} \Big(\bigcup_{j \in \{1, \ldots, H(l)\}} \{(g_+(l, j), (-\delta_{il})_{1 \leq i \leq r})\} \cup \{(g_-(l), (+\delta_{il})_{1 \leq i \leq r})\} \Big).$$

Then, if S is lossless and repetition-free, the determinacy of S can be decided by testing whether there are vectors in $\mathbf{R}(\mathbf{W}')$ with $(r+l)$-th and $(r+l')$-th coordinate $\geqq 1$ and $(l, l') \in \rho$.

Now assume that in the partition $\mathcal{N}$ of N there is more than one element of infinite cardinality and that S is lossless, repetition-free, locally finite and also compact. Let exactly $N_m, \ldots, N_r$ be infinite and let $k := |\mathbf{K}'|$. If there is an l, $m \leqq l \leqq r$, such that there are more than k operations $o(i) \in \mathrm{Op}$ with $i \in N_l$ and $R(i) \cap \mathbf{K}' \neq \emptyset$ then there are $i_1, i_2 \in N_l$ with $i_1 \neq i_2$ and $R(i_1) \cap R(i_2) \neq \emptyset$ and S is not determinate. We thus can assume that for each N_l, $l \in_r N$, there are at most k operations $o(i) \in \mathrm{Op}$ with $i \in N_l$ and $R(i) \cap \mathbf{K}' \neq \emptyset$; let $K_R(l)$ denote the set of these i. By local finiteness there can be at most finitely many operations $o(i') \in \mathrm{Op}$ with $i' \in N_{l'}$ such that there is an $i \in K_R(l)$, $(i, i') \in K'$ and $D(i') \cap R(i) \neq \emptyset$. Assume that there are infinitely many operations $o(i) \in \mathrm{Op}$ with $i \in N_l$ and $D(i) \cap \mathbf{K}' \neq \emptyset$; let $K_D(l)$ denote the set of those i. Then there exists at least one l', $m \leqq l' \leqq r$, with $K_R(l')$ infinite contrary to our assumption. Thus also $K_D(l)$ is finite for all $l \in_r N$. Thus $\{ o(i) \in \mathrm{Op} \mid (D(i) \bigcup R(i)) \cap \mathbf{K}' \neq \emptyset \}$ is finite and also K' is finite. If K' can be effectively constructed and if we have a decision algorithm for each N_l, $m \leqq l \leqq r$, then we can effectively construct the above defined relation ρ and decide the determinacy of S. We first present a definition and then state these considerations as a theorem.

DEFINITION 1.12 : An SVAS S is *suitably presented* if:

(1) each infinite element N_l of the partition $\mathcal{N}$ of N is decidable and

(2) if K' is finite then K' can be effectively constructed.

THEOREM 1.1: *Let* S *be a lossless, repetition-free locally finite and suitably presented SVAS and let* $\mathcal{N}$ *be the partition of* N *associated with* S*:*

(*a*) *If there is only one element of infinite cardinality in* $\mathcal{N}$ *then it is decidable whether* S *is determinate.*

(*b*) *If there is more than one element of infinite cardinality in* $\mathcal{N}$ *and* S *is compact, too, then it is decidable whether* S *is determinate.*

Thus our extension of the main theorem of Karp and Miller [9] essentially results out of the fact, that conditions, which one would impose on unbounded schemata out of motivations external to schemata theory, permit one to reduce some aspects of the decidability of their determinacy to those of bounded schemata.

2. THE PARALLEL TIME COMPLEXITY OF PARALLEL COMPUTATIONS

The main reason for considering parallel algorithms is the hope that a parallel execution of a parallel algorithm will take significantly less time than a sequential execution. Therefore one is interested, given a parallel computation, to devise a

more parallel computation which computes the same results in less parallel time. It is this line of investigation which Keller has followed in [11, 12]. We have looked at things differently by trying to answer questions like: What is a reasonable complexity measure for parallel computations of a UKMS? What is the (minimal) parallel execution time for an algorithm represented by an SVAS? Are there kinds of sequential algorithms which are good starting points for designing parallel algorithms? Most of our answers are different from the ones Keller has given in [11, 12] because we are looking at these questions more from the viewpoint of the algorithms than from the viewpoint of the schemata.

We commence by proposing a complexity measure for parallel computations of a UKMS. Its definition is so simple that we hope to have got hold of salient features of the natural complexity measure which is intuitively used for parallel computations. Before we give the definition we explain the idea behind it and introduce some notation.

A complexity measure for the execution time of a parallel computation must take into account that several operations might be executed simultaneously and that the duration of an operation must no more be measured from the moment of the termination of the operation terminated at last. Because UKMS provide for initiation and termination symbols and because of their generality permit the representation of a very big number of algorithms we define our complexity measure on all finite computations of a UKMS and their prefixes. To this aim a strictly positive map $v : N \to N_+$, the execution time, must be given; $v(i)$ specifies the execution time of the i-th operation $o(i)$. We measure the moment of termination of an operation from the moment of its initiation. We consider a sequence consisting of initiation symbols only as contemporaneous, and we assign the latest of the moments of termination to a sequence consisting of termination symbols only. The latter prescription is the main difference between the complexity measure of [11, 12] and our's; for another approach see Keller's thesis [10]; also Keller's complexity measure is defined in a more general setting that we do not deem necessary for our investigations.

For the formal definition of our complexity measure we need the following notation. Let $x\underline{o(i)} \in P(S)$ be a prefix of a computation of a UKMS S. Then we write $i(x)$ for the substring $y\overline{o(i)}$ of x in which the indicated occurence of $o(i)$ is the initiation symbol corresponding to the indicated occurrence of the termination symbol $\underline{o(i)}$ in $x\underline{o(i)}$.

DEFINITION 2.1: Let S be a UKMS and $v : \mathrm{Op} \to N_+$ be a strictly positive map, the execution time. Then a *parallel complexity measure* $m : P(S) \to N$ is recursively defined by

$$m(\lambda) := 0 \text{ where } \lambda \text{ is the null string;}$$

$$m(x\overline{o(i)}) := m(x);$$
$$m(x\underline{o(i)}) := \max\{m(x),\ m(i(x))+v(i)\}.$$

Example 2.1: Consider the computation $x := \overline{o(1)\,o(3)}\,\underline{o(1)}\,\overline{o(2)}\,\underline{o(3)\,o(2)}$. We have $m(x)=\max\{v(1)+v(2),\ v(3)\}$. Now consider the computation $x' := \overline{o(1)\,o(3)}\,\underline{o(1)}\,\overline{o(2)}\,\underline{o(2)\,o(3)}$. Again we have $m(x')=\max\{v(1)+v(2),\ v(3)\}$; thus $m(x)=m(x')$. If S is an SVAS then $x\in C(S)$ iff $x'\in C(S)$; thus our complexity measure characterizes a class of computations and not only a single computation.

After having defined a parallel complexity measure on the prefixes of the computations of a UKMS we now present an iterative algorithm which finds a fastest computation of a schema for this complexity measure. This algorithm will answer our second question – what is the parallel execution time for an algorithm given by an SVAS? – because in our view the parallel execution time of an algorithm given by a schema is the execution time of the fastest computation of this schema. This seems to be a reasonable viewpoint if one observes that in general in every parallel execution of an algorithm one has to choose between different parallel computations of the algorithm and one often chooses the one which is fastest.

In order to simplify the otherwise quite difficult problem we make two additional assumptions:

(1) that the control of the schema can be specified in the form of a precedence graph, *cf.* [5], and

(2) that the precedence graph is a dag (directed acyclic graph).

The first assumption makes it necessary to construct a precedence graph from the control specified in the manner of definition 1.1. Of course the precedence graph embodies only part of the properties of the control, namely those that are relevant to constructing the fastest computation, which is exactly what we want. For certain determinate UKMS the construction of the precedence graph $G=(V, E)$ goes as follows. The (possibly infinite) set of vertices V consists of exactly those operations $o(i)\in \mathrm{Op}$ which are terminated in the computations of the schema. The roots of G are those vertices $o(i)\in V$ for which $t(q_0, \overline{o(i)})$. Furthermore $(o(i), o(j))\in E$ iff

$$(\exists y\in O^*)\quad (\exists v\in \underline{O}^*)\quad (\exists w\in \overline{O}^*)\quad (\exists z\in \hat{O})\quad (y\underline{o(i)}\ vw\overline{o(j)}z\in P(S))$$
$$\wedge(\forall y'\in O^*)\quad (\forall v'\in \underline{O}^*)\quad (\forall w'\in \overline{O}^*)\quad (\forall z'\in \hat{O})\quad (y'\overline{o(j)}\,w'\,v'\,\underline{o(i)}\,z')\notin P(S).$$

It might be necessary to modify the above construction for certain UKMS.

The second assumption is necessary to reduce the problem to a scheduling problem (in the sense of [5]).

From now on we base our considerations only on the precedence graph G and no more on the control specified in the manner of definition 1.1. Before we present our iterative algorithm we informally described the idea behind it. The input to the algorithm are the precedence graph G and the execution times $v(i)$ of the operations $o(i) \in V$; the output is a fastest computation x of the schema underlying G.

In a procedure called "NEXT" those operations, which are going to be treated in the current iteration, are read into the set C. Like in critical path algorithms the earliest possible initiation times $e(i)$ of all $o(i) \in C$ are computed. Then the set T of operations which have not yet been terminated and the set I of operations which have not yet been initiated are augmented by the elements of C. We assume that the precedence graph has been completed by a START and a HALT operation in the usual way. After the obvious initialization of the sets Y, T and I the procedure NEXT is called for the first time to input the first operations.

The main part of the algorithm consists of a repeat loop. First the set T' of all operations which have to be terminated in this iteration is constructed as $T' := \{o(i) \in T \mid o(i)$ has already been initiated and

$$m(x\underline{o(i)}) \leqq \min\{e(i') \mid o(i') \in I\}\}.$$

After the termination of the operations in T' the procedure NEXT is called again because new operations might be enabled. Then the set I' of all operations which have to be initiated in this iteration is constructed as

$$I' := \{o(i) \in I \mid e(i) = \min\{e(i') \mid o(i') \in I\}\}.$$

This finishes the contents of the repeat loop. This loop is executed until $I = \emptyset$.

We now present the algorithm that constructs a fastest computation of a UKMS whose control is specified in the form of a directed acyclic precedence graph; several obvious improvements are not applied to the algorithm for the sake of simplicity.

ALGORITHM 2.1:

```
procedure NEXT;
begin   L := C;
        C := {o(i) ∈ V − Y | ∅ ≠ {o(i') ∈ V | (o(i'), o(i)) ∈ E} ⊂ Y};
        for o(i) ∈ C do e(i) := max{e(i') + v(i') | (o(i'), o(i)) ∈ E};
```

```
        Y:=Y∪C;
        T:=T∪C;
        I:=I∪C;
end;
begin comment initialization; Y:=C:={START};
                              x:=λ;
                              T:=I:=∅;
                              e(START):=0;
                              NEXT;
      comment main part;
      repeat T':={o(i)∈T | o(i)∈x and
                  m(xo(i))≦min{e(i') | o(i')∈I}};
             T:=T−T';
             while T'≠∅ do begin take o(i) out of T';
                                 x:=xo(i);
                           end;
             NEXT;
             I':={o(i)∈I | e(i)=min{e(i') | o(i')∈I}};
             I:=I−I';
             repeat take o(i) out of I';
                    x:=xo(i);
             until I'=∅;
      until I=∅;
end.
```

We now prove the correctness of algorithm 2.1 and calculate its time complexity.

The following lemma is necessary because the output of our algorithm is not the input to a list schedule but a computation of the schema underlying the precedence graph.

LEMMA 2.1: *Algorithm* 2.1 *does not permit the initiation of an operation before all of its predecessors in the precedence graph have been terminated, i.e. every prefix of the computation constructed by the algorithm respects the precedence graph.*

Proof: Let $o(i), o(k) \in V$ with $(o(k), o(i)) \in E$ and $o(k) \neq$ START. Because we have assumed that $v(i) > 0$ for all $o(i) \in$ Op we certainly have $e(k) < e(i)$, and, according to the construction of NEXT and I', the operation $o(k)$ has been initiated before the operation $o(i)$. We still have to show that $m(x\underline{o(k)}) \leqq e(i)$ where x is the prefix constructed by algorithm 2.1 after which $o(k)$ is terminated. But this follows from the structure of the outer repeat loop, which causes $o(k)$ to be terminated before $o(i)$ is initiated.

LEMMA 2.2: *At each call of the procedure* NEXT *the set C is augmented by at least one element.*

Proof: Because of the construction of G each element of V-{START} can be reached from START. The assertion now follows immediately from the structure of Y.

LEMMA 2.3: *At each execution of the outer repeat loop at least one initiation symbol is added to the already constructed prefix of the computation.*

Proof: This follows immediately from the construction of I' and the finiteness of I.

COROLLARY 2.1: *The outer repeat loop is executed at most* $|V|$ *times.*

COROLLARY 2.2: *Algorithm 2.1 finds a computation x for the UKMS underlying the precedence graph G after at most* $|V|$ *passes through the outer repeat loop.*

Proof: Immediate from lemma 2.1 and the definition of the HALT operation.

THEOREM 2.1: *For the computation x of corollary 2.2 we have: if x' is another computation for the schema underlying the precedence graph G then* $m(x') \geqq m(x)$.

Proof: According to the construction of T' we have $m(x) \leqq e(\text{HALT})$. Actually we even have $m(x) = e(\text{HALT})$ from the definition of e. Therefore $m(x') \geqq m(x)$, because $m(x') < m(x)$ would contradict the definition of e.

COROLLARY 2.3: *Algorithm 2.1 constructs a fastest computation of the schema underlying the precedence graph G.*

LEMMA 2.4: *In each pass through the outer repeat loop, with the exception of the first one, algorithm 2.1 adds at least one termination symbol to the already constructed prefix of the computation x of corollary 2.2.*

Proof: It follows from the construction of T' that only operations from $T-I$ are terminated. Because $T=I$ in the first pass no operations are terminated.

Consider a pass which is not the first one and in which no operations are terminated. Because of lemma 2.3 at least one operation is initiated in that pass, say $o(i)$. Let $\underline{o(j)}$ be the termination symbol for which a string $yo\underline{(j)}$ yields the maximum in the computation of $e(i)$. Because of lemma 2.1 the operation $o(j)$ then has already been terminated in an earlier pass, in which among others the operation $o(k)$ has been initiated. Thus $m(x\underline{o(j)}) \leqq e(k) < e(i)$ which contradicts $m(x\underline{(o(j)}) = e(i)$.

Because of corollary 2.2, algorithm 2.1 can be considered n^3, where $n = |V|$ for finite V. This is interesting because the analogous scheduling problem for a

bounded number of parallel processors has been proven NP-complete in [18]. It is furthermore immediate that algorithm 2.1 is really suited for precedence graphs with an unbounded number of operations, because it can be executed simultaneously with the execution of the operations.

We now consider the execution of algorithm 2.1 on the precedence graph of a celebrated algorithm of complexity theory, namely the algorithm of Strassen for multiplication of 2×2-matrices in the version of Winograd, *cf.* [1]. In appendix II we present the precedence graph of the Strassen-Winograd algorithm and a "dump" of the execution of algorithm 2.1 on this precedence graph containing the contents of C, T, I, T' and I' each time they are assigned some value and also denoting each call of the procedure NEXT.

THEOREM 2.2: *Algorithm* 2.1 *finds a parallel computation of the schema underlying the Strassen-Winograd algorithm for multiplication of* 2×2*-matrices which is faster by the execution time of two parallel additions than the parallel computation suggested by the sequential execution of the Strassen-Winograd algorithm.*

In practice it is intractable to find the fastest parallel computation by enumeration of all parallel computations because even a small algorithm like the Strassen-Winograd algorithm, which consists only of 21 different operations, might have an enormous number of parallel computations, e.g. the Strassen-Winograd algorithm has more than 1.4×10^{10} different parallel computations, as can be easily seen. In this fact we see the justification for the development of algorithm 2.1.

3. L SYSTEMS AS MODELS FOR PARALLELISM IN ALGORITHMS

In this section we interpret L systems as models for parallel algorithms and try to obtain decidability results for the theory of parallel algorithms from the known decidability results of the theory of L systems. It should be noted that our approach is just a continuation of a philosophy already expressed in [8] where, with the simulation of the firing squad synchronization problem as a PD2IL scheme, [*cf.* 8, p. 278], it was proven that L systems are able to model the essential features of a parallel algorithm. In what follows we propose to systematically consider L systems as models of the control and data structure of parallel algorithms and first demonstrate our approach in two examples.

Example 3.1: Consider multiplication of square matrices of size 2^n, $n \in N$. Let (a_{ij}), (b_{ij}) be the input matrices; then the obvious algorithm forms $c_{ij} = \sum_k a_{ik} b_{kj}$ where (c_{ij}) is the output matrix. This algorithm can be parallelized by

simultaneously executing the 2^{3n} multiplications $a_{ik}\,b_{kj}$ and then executing the $2^{2n}(2^n-1)$ additions by the divide and conquer method; this takes n steps of parallel additions. The following $D0L$ system can be regarded as a representation of the obvious parallel algorithm for multiplication of matrices of size 2^n:

$$G_n=\langle \Sigma_n,\ P_n,\ \omega\rangle;$$

$$\Sigma_n=\{\omega\}\cup \bigcup_{1\leqq i\leqq n+1}\ \bigcup_{1\leqq j\leqq 2^{3n-i+1}} \{A_{ij}\};$$

$$P_n=\{\langle \omega,\ A_{11}\ldots A_{1,\,2^{3n}}\rangle \cup \bigcup_{1\leqq j\leqq 2^{2n}} \langle A_{n+1,\,j},\ \lambda\rangle$$

$$\cup \bigcup_{1\leqq i\leqq n}\Big(\bigcup_{j\in O_i}\langle A_{ij},\ \lambda\rangle \cup \bigcup_{j\in E_i}\langle A_{ij},\ A_{i+1,\,j/2}\rangle\Big)\}$$

where O_i, resp. E_i, denotes the set of odd, resp. even, integers between 1 and 2^{3n-i+1}. The elements of $L_m(G_n)$ represent the operations executed by the algorithm in the m-th parallel time unit. In the first time unit, the 2^{3n} multiplications are executed, which are represented by the symbols A_{1j}, $1\leqq j\leqq 2^{3n}$. In the next time unit, $2^{2n}(2^{n-1})$ additions are executed, which are represented by the symbols A_{2j}, $1\leqq j\leqq 2^{3n-1}$. In the i-th time unit, $2\leqq i\leqq n+1$, 2^{3n-i+1} additions are executed which are represented by the symbols A_{ij}, $1\leqq j\leqq 2^{3n-i+1}$. In the $(n+2)$th time unit, no operation is executed because the result is already computed, consequently $L_{n+2}(G)=\{\lambda\}$.

Example 3.2: Consider the knapsack problem in the form $\sum_{1\leqq i\leqq\infty} a_i x_i\leqq b$; usually only the finite version $\sum_{1\leqq i\leqq n} a_i x_i\leqq b$ is considered. A trivial enumeration algorithm enumerates all elements of $\{0,\ 1\}^*$, where an element of $\{0,\ 1\}^*$ represents a possible value of an initial subsequence of $(x_i)_{1\leqq i\leqq\infty}$, and checks, which values satisfy the above inequality. The following $D0L$-system represents that part of the algorithm that constructs the elements of $\{0,\ 1\}^*$:

$$G=\langle \Sigma,\ P,\ \omega\rangle;$$

$$\Sigma=\langle \omega,\ 0,\ 1,\ 0',\ 1'\rangle;$$

$$P=\{\langle \omega,\ 01\rangle,\ \langle 0,\ 0'01\rangle,\ \langle 1,\ 1'01\rangle,\ \langle 0',\ 0'\rangle,\ \langle 1',\ 1'\rangle\}.$$

In the following we explain why $L_i(G)$ can be identified with the set of all subsets of $\{0,\ 1\}^i$. The general rule is: $L_i(G)$ represents those strings in $\{0,\ 1\}^*$ which are produced by the algorithm in the i-th parallel time unit. The algorithm starts on the empty string λ, which can be considered as being produced in the zeroth time unit and which is represented by the axiom ω. In the first time unit, the strings 0 and 1 are produced, and in the i-th time unit all elements of $\{0,\ 1\}^i$ are

produced. Consequently $L_i(G)$ should represent all elements of $\{0, 1\}^i$ and it does so in the following way: an element of $\{0, 1\}^i$ is represented by a certain substring of the only string in $L_i(G)$, which we also denote by ω_i, where it should be noted that $\omega_i \in \{0, 1, 0', 1'\}^*$ for $i \geqq 1$. Now there is a map f which bijectively maps those initial sections ν of some ω_i that end with an element of $\{0, 1\}$, onto the elements of $\{0, 1\}^i$. In order to define f we introduce the map u: $\{0', 1'\}^* \to \{0, 1\}^*$ which is the unique extension to a monoid morphism of the map $\{0', 1'\} \to \{0, 1\}$ given by $0' \to 0$, $1' \to 1$; then the mentioned map f can be defined recursively as follows:

(*a*) let $\nu = \alpha' \delta$, where $\alpha' \in \{0', 1'\}^*$ and $\delta \in \{0, 1\}$; then $f(\nu) = u(\alpha')\delta$;

(*b*) let $\nu = \alpha\gamma'\delta$, where $\alpha \in \{0', 1', 0, 1\}^* \{0, 1\}^2$, $\gamma' \in \{0', 1'\}^m$ and $\delta \in \{0, 1\}$; let $f(\alpha) = \beta_1 \beta_2$, where $|\beta_2| = m+1$; then $f(\nu) = \beta_1 u(\gamma')\delta$;

(*c*) let $\nu = \alpha\delta_1\delta_2$, where $\alpha \in \{0', 1', 0, 1\}^*$ and $\delta_1, \delta_2 \in \{0, 1\}$; let $f(\alpha\delta_1) = \beta\delta_1$, then $f(\nu) = \beta\delta_2$.

We now illustrate this map in some examples. Consider $\omega_2 = 0'011'01$. According to (*a*) $f(0'0) = 00$; from (*c*) $f(0'01) = 01$; from (*b*) $f(0'011'0) = 10$; from (*c*) $f(0'011'01) = 11$. Consider $\omega_3 = 0'0'011'011'0'011'01$. From (*a*) $f(0'0'0) = 000$; from (*c*) $f(0'0'01) = 001$; from (*b*) $f(0'0'011'0) = 010$; from (*c*) $f(0'0'011'01) = 011$; from (*b*) $f(0'0'011'011'0'0) = 100$; from (*c*) $f(0'0'011'011'0'01) = 101$; from (*b*) $f(0'0'011'011'0'011'0) = 110$; from (*c*) $f(\omega_3) = 111$. As is easily seen the last equality generalizes: $f(\omega_i) = 1^i$.

In the following we define two classes of UKMS each containing respectively one of the two examples above. They have the property, that either their operations or their intermediate results can be characterized by an L system in such a way that certain decidability results for L systems can be directly applied to the UKMS. This is of course only a very first step in trying to establish a connection between the theory of L systems and the theory of parallel algorithms but the fact that a connection can be established so easily bolsters the hope that a deeper connection can be found in further work.

DEFINITION 3.1: Let $S = (\mathrm{Op}, T, D, R, H)$ be a UKMS, $v : N \to N_+$ the execution time of the operations of S and $G = \langle \Sigma, P, \omega \rangle$ a $0L$ system. For all computations x in $C(S)$ let m'_x be that subset of $F(N) \times N$ [where $F(N)$ denotes the set of finite subsets of N] with the property that $(\{j_1, \ldots, j_k\}, i) \in m'_x$ iff $\{o(j_1), \ldots, o(j_k)\}$ is exactly the set of operations executed in the computation x during the i-th time unit. Then we say that S is a *G control UKMS* (short: GCUKMS) iff there is a map $f_c : \Sigma^* \to F(N)$ whose restriction to $L(G)$ is injective and which has the properties that for any $\mathbf{n} \in F(N)$ we can effectively

construct $f_c^{-1}(\mathbf{n})$ and

$$(\exists x \in C(S)) \quad ((f_c(\alpha), i) \in m'_x) \quad \Leftrightarrow \quad \alpha \in L_i(G).$$

Similarly define $R'_x \subset F(N) \times N$ such that $(\{j_1, \ldots, j_k\}, i) \in R'_x$ iff $\{j_1, \ldots, j_k\}$ is exactly the set of memory locations into which results are stored by the operations of x terminating in the i-th time unit. Then we say that S is a *G data UKMS* (short: GDUKMS) iff there is a mapping $f_d : \Sigma^* \to F(N)$ whose restriction to $L(G)$ is injective and which has the properties that for any $\mathbf{n} \in F(N)$ we can effectively construct $f_d^{-1}(\mathbf{n})$ and

$$(\exists x \in C(S)) \quad ((f_d(\alpha), i) \in R'_x) \quad \Leftrightarrow \quad \alpha \in L_i(G).$$

We now informally describe what is behind the above definition. In both cases, when S is a GCUKMS or a GDUKMS, important aspects of computations of S can be characterized by properties of the $0L$ system G. When S is a GCUKMS then the parallel execution of operations in computations of S is mirrored by the elements of $L(G)$; when S is a GDUKMS then the storage of results of operations in computations of S is mirrored by the elements of $L(G)$. Example 3.1 is an example for a GCUKMS and example 3.2 is an example for a GDUKMS. In both cases the algorithm has not been explicitly given in schema form because this would not have given any further insight. Also in both cases the $0L$-system G is a $D0L$-system which is tantamount to a considerable simplification in specifying the maps f_c and f_d, respectively, but which is paid for by restricting the number of computations of the GCUKMS or GDUKMS; this is a disadvantage more from the viewpoint of decidability than from the viewpoint of parallelism.

THEOREM 3.1 : *Let S be a GCUKMS and $j_1, \ldots, j_n \in N$ such that $o(j_1), \ldots, o(j_n) \in \mathrm{Op}$. Then it is decidable whether there is a computation $x \in C(S)$ and a natural number $i \in N$ such that $(\{j_1, \ldots, j_n\}, i) \in m'_x$; i.e. given any finite set of operations of S it is decidable, whether exactly these operations are executed simultaneously during some time unit in any computation of S.*

Proof: Let $G = \langle \Sigma, P, \omega \rangle$ be the underlying $0L$-system such that $f_c : \Sigma^* \to F(N)$ makes S a GCUKMS. If $|f_c^{-1}(\{j_1, \ldots, j_n\})| \neq 1$ then certainly there is no computation $x \in C(S)$ such that $\{o(j_1), \ldots, o(j_n)\}$ is the set of operations executed simultaneously in any time unit of the execution of x. If $|f_c^{-1}\{(j_1, \ldots, j_n\})| = 1$ then

$$(\exists x \in C(S)) \quad (\exists i \in N) \quad ((\{j_1, \ldots, j_n\}, i) \in m'_x) \quad \Leftrightarrow \quad f_c^{-1}(\{j_1, \ldots, j_n\}) \in L(G).$$

But as the membership problem for $0L$-systems is decidable according to [8, p. 76] the above problem is decidable, too.

In the same easy way one obtains the following analogous result for GDUKMS.

COROLLARY 3.1: *Let S be a GDUKMS and* $j_1, \ldots, j_n \in N$. *Then it is decidable whether*

$$(\exists x \in C(S)) \quad (\exists i \in N) \qquad ((\{j_1, \ldots, j_n\}, i) \in R'_x);$$

i.e. given any finite set of memory locations, it is decidable, whether there is a computation x *in* $C(S)$ *such that, for some time unit in the execution of* x, *the set of memory locations into which results are stored during this time unit is exactly the given set.*

We consider theorem 3.1 as a strong hint that the theory of L systems is not only applicable to biological and related phenomena but also to the theory of parallel programs.

We conclude with a bibliographical remark. The first paper known to us, in which a formal connection between a model for parallel processes and grammars, which are very similar to L systems with interaction, is established, is [13]; however, it is not mentioned there that the grammars investigated are similar to L systems with interaction. Also in [17] the theory of parallel algorithms and the theory of L-systems are presented in one common volume though apparently no formal connection had been established.

ACKNOWLEDGEMENT

I gratefully acknowledge the constant and untiring support of Prof. Dr. Erwin Engeler during all of this work. It is based on several ideas presented to me by him. I am, however, the only one to be blamed for any shortcomings or errors in their carrying-out.

I also warmly thank the anonymous referee. He has not only eliminated several mistakes and proposed (among many other things) a profound modification of the first section that has resulted in a considerable improvement but has done all this in a very cooperative and extraordinarily competent way.

Further thanks are due to Prof. A. C. Shaw, Dr. H.-H. Nägeli and P. Horák for linguistic help.

REFERENCES

1. A. V. AHO, J. E. HOPCROFT and J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Exercise 6.5, p. 247, Addison-Wesley, Reading, Mass., 1974.
2. R. ALBRECHT, *Zur Struktur von Informationssystemen*, in *Automatentheorie und formale Sprachen*, Oberwolfach, 1969, J. DÖRR and G. HOTZ Eds., *Berichte aus dem mathematischen Forschungsinstitut Oberwolfach*, Vol. 3, 1970, pp. 493-505, Bibliographisches Institut, Mannheim.

3. B. BUCHBERGER, *Computer-Trees and Their Programming*, Troisième Colloque de Lille sur les Arbres en Algèbre et en Programmation, 1978.
4. B. BUCHBERGER and J. FEGERL, *Ein universeller Modul zur Hardware-Implementierung von Rekursionen*, Laboratory Note, Universität Linz, 1977.
5. E. G. COFFMAN (ed.), *Computer and Job Shop Scheduling Theory*, Wiley-Interscience, New York, 1976.
6. R. W. CONWAY, W. L. MAXWELL and L. W. MILLER, *Theory of Scheduling*, Addison-Wesley, Reading, Mass., 1967.
7. G. GATI, *Aspects of Unbounded Parallelism*, in *Theoretical Computer Science*, 3rd G.I. Conference, Darmstadt, 1977, H. TZSCHACH, H. WALDSCHMIDT and H. K.-G. WALTER, Eds., Lecture Notes in Computer Science, Vol. 48, pp. 292-305, Springer, Berlin, 1977.
8. G. T. HERMAN and G. ROZENBERG, *Developmental Systems and Languages*, North-Holland-American Elsevier, Amsterdam, 1975.
9. R. M. KARP and R. E. MILLER, *Parallel Program Schemata*, J. Comput. System Sc., Vol. 3, 1969, pp. 147-195.
10. R. M. KELLER, *Closures of Parallel Program Schemata*, Unpublished thesis, University of California, 1970.
11. R. M. KELLER, *On Maximal Parallel Schemata*, in *Conference Record of 11th Annual Symposium on Switching and Automata Theory*, I.E.E.E., New York, 1970, pp. 32-50.
12. R. M. KELLER, *Parallel Program Schemata and Maximal Parallelism*, J. Assoc. Comput. Mach., Vol. 20, 1973, pp. 514-537 (part I) and Vol. 20, 1973, pp. 696-710 (part II).
13. H. KOPP, *Bemerkungen zum Deadlockproblem in parallelen Prozessen*, Unpublished report, Tagung über Automatentheorie und formale Sprachen, Mathematisches Forschungsinstitut Oberwolfach, 1975.
14. F. SCHWENKEL, *Zur Theorie unendlicher Parallelprozessoren*, in *G.I.-4. Jahrestagung*, Berlin, 1974, D. SIEFKES, éd., (im Auftrag der G.I.), Lecture Notes in Computer Science, Vol. 26, pp. 355-364, Springer, Berlin, 1975.
15. H. SULLIVAN and T. R. BASHKOW, *A Large Scale, Homogeneous, Fully Distributed Parallel Machine*, I. Comp. Arch. News, Vol. 5, 1977, pp. 105-117.
16. H. SULLIVAN, T. R. BASHKOW and D. KLAPPHOLZ, *A Large Scale, Homogenous, Fully Distributed Parallel Machine*, II. Comp. Arch. News, Vol. 5, 1977, pp. 118-124.
17. J. T. TOU (ed.), *Advances in Information Systems Science*, Vol. 6, 1976, Plenum Press, New York.
18. J. D. ULLMAN, *NP-Complete Scheduling Problems*, J. Comput. System Sc., Vol. 10, 1975, pp. 384-393.

APPENDIX I

We here prove corollary 1.1 in the following way: corollary 1.1 is part of corollary A.2 below which is deduced from a sequence of other results in a manner similar to the proof of corollary 2.2 of [9].

THEOREM A.1: *Let S be a lossless SVAS; then S is determinate if and only if the following condition holds for every interpretation I:*

(*) *If* u *is in* O^* *and* σ, $\pi \in O$ *such that* $\langle \alpha_0 \cdot u\sigma\pi \rangle$ *and* $\langle \alpha_0 \cdot u\sigma\pi \rangle$ *then* $\alpha_0 \cdot u\pi\sigma = \alpha_0 \cdot u\sigma\pi$.

This theorem will be proved with the help of a series of lemmas.

LEMMA A.1: *Condition* (*) *of theorem A.1 holds for every interpretation iff it holds for every one-one interpretation.*

Proof: Follows from lemma 1.1.

LEMMA A.2: *Let S be a lossless SVAS, $I \in I'(S)$ and $\alpha \in Z(I)$. Then, for each pair $(o(i).\ o(j)) \in \mathrm{Op} \times \mathrm{Op}$, with $i \neq j$:*

(*a*) *if $\langle \alpha \cdot \overline{o(i)o(j)} \rangle$ and $\alpha \cdot \overline{o(j)o(i)}$ then*

$$\alpha \cdot \overline{o(i)o(j)} = \alpha \cdot \overline{o(j)o(i)};$$

(*b*) *if $\langle \alpha \cdot \overline{o(i)}\ \underline{o(j)} \rangle$ and $\langle \alpha \cdot \underline{o(j)}\ \overline{o(i)} \rangle$ then*

$$\alpha \cdot \overline{o(i)}\ \underline{o(j)} = \alpha \cdot \underline{o(j)}\ \overline{o(i)}$$

iff:

(i) $R(j) \cap D(i) = \emptyset$ *or*

(ii) $o(j)$ *is a repetition, i. e. does not change the contents of any element of* $R(j)$;

(*c*) *if $(\alpha \cdot \underline{o(i)o(j)})$ and $(\alpha \cdot \underline{o(j)o(i)})$ then*

$$\alpha \cdot \underline{o(i)o(j)} = \alpha \cdot \underline{o(j)o(i)}$$

iff $R(i) \cap R(j) = \emptyset$.

Proof: Same as proof of lemma 2.3 of [9].

LEMMA A.3: *Let S be a lossless SVAS and $I \in I'(S)$. Let $v \in O^*$, $\sigma, \pi \in O$ such that $\alpha_0 \cdot v\,\sigma\pi = \alpha_0 \cdot v\,\pi\sigma$. Then, for any $w \in \hat{O}$:*

(*a*) $v\,\sigma\pi\,w \in C(S, I)$ *iff* $v\,\pi\sigma\,w \in C(S, I)$;

(*b*) $(\forall i \in N)\ (\Omega(v\,\sigma\pi\,w, i) = \Omega(v\,\pi\sigma\,w, i))$.

Proof: Same as proof of lemma 2.4 of [9].

LEMMA A.4: *Let S be an SVAS, $I \in I(S)$, $u \in O^*$, $w \in \hat{O}$ and $\sigma \in O$. If $\langle \alpha_0 \cdot u\,\sigma \rangle$ and $uw \in C(S, I)$ then $\sigma \in w$.*

Proof: Follows from definition 1.4, condition (3).

Proof of theorem A.1: Let $I \in I'(S)$ such that condition (*) is satisfied. Suppose that x, y are in $C(S, I)$ with $\Omega(x, i) \neq \Omega(y, i)$ for some i. We shall show that for all $n \leqq |x|$ (for all n if $x \in O^\infty$) there is a $z(n) \in C(S, I)$ such that:

(i) $(\forall i \in N)\ (\Omega(z(n), i) = \Omega(y, i))$ and

(ii) ${}_n z(n) = {}_n x$.

It will then follow that $\Omega(x, i)=\Omega(y, i)$ for all i, giving a contradiction. The proof is by induction. Setting $z(0)=y$ we have the result for $n=0$. Let us assume that the result holds for $n=k$ and that $|x| \geqq k+1$. Then $(\alpha_0 \cdot ({}_k x)) \cdot x_{k+1}$ is defined, or, equivalently, $(\alpha_0 \cdot ({}_k z(k))) \cdot x_{k+1}$ is defined. For brevity, let p denote ${}_k z(k)$. By lemma A.4, $z(k)$ can be written as $z(k)=pvx_{k+1}s$ where $v \in O^*$, $s \in \hat{O}$ and $x_{k+1} \notin v$. Assume that x_{k+1} is $\overline{o(j)}$ or $\underline{o(j)}$ with $j \in N_l$. We shall show that $z(k)$ can be chosen in such a way that in addition v contains no $\overline{o(j')}$ and no $\underline{o(j')}$ with $j' \in N_l$. Suppose that v is of the form $v=v'v''$ with

$$v'=w'\overline{o(j')}\pi' u'\underline{o(j')}, \qquad v''=w''\pi'',$$
$$j' \in N_l, \qquad \pi', \pi'' \in O, \qquad w', w'', u' \in O^*,$$

the indicated occurrences of $\overline{o(j')}$ and $\underline{o(j')}$ in v' are corresponding initiation and termination symbols and v'' contains no $\overline{o(j'')}$ and no $\underline{o(j'')}$ with $j'' \in N_l$. As π'' is not associated with N_l, we have $\langle((\alpha_0 \cdot pv'w'') \cdot x_{k+1}) \cdot \pi''\rangle$; this is trivial if $x_{k+1}=\underline{o(j)}$ and follows from definition 1.8, $(1d)$, if $x_{k+1}=\overline{o(j)}$. By condition (*) we have

$$\alpha_0 \cdot pv'w''x_{k+1}\pi''=\alpha_0 \cdot pv'w''\pi''x_{k+1}.$$

By induction based on this argument we have

$$\alpha_0 \cdot pv'x_{k+1}v''=\alpha_0 \cdot pv'v''x_{k+1}.$$

We now show that $\langle\alpha_0 \cdot pw'\pi'u'\overline{o(j')}\rangle$ and that

$$\alpha_0 \cdot pw'\pi'u'\overline{o(j')}=\alpha_0 \cdot pw'\overline{o(j')}\pi'u'.$$

This follows if we can show that $\langle\alpha_0 \cdot pw'\pi'\overline{o(j')}u'\rangle$; because then condition (*) implies $\alpha_0 \cdot pw'\pi'\overline{o(j')}u'=\alpha_0 \cdot pw'\overline{o(j')}\pi'u'$ and, as π' is arbitrary, we can proceed by induction and "slide" $\overline{o(j')}$ past each element of u'. In order to show that $\langle\alpha_0 \cdot pw'\pi'\overline{o(j')}u'\rangle$ we have to consider three cases, namely (i) π' is not associated with N_l, (ii) $\pi'=\overline{o(j'')}$ with $j'' \in N_l$ and (iii) $\pi'=\underline{o(j'')}$ with $j'' \in N_l$. In each case it follows from definition 1.8 that $\langle\alpha_0 \cdot pw'\pi'\overline{o(j')}u'\rangle$ and we can conclude by the remark above that

$$\alpha_0 \cdot pw'\overline{o(j')}\pi'u'\underline{o(j')}=\alpha_0 \cdot pw'\pi'u'\ \overline{o(j')}\,\underline{o(j')}.$$

If $x_{k+1}=\overline{o(j)}$ suppose furthermore that $z(k)=pvx_{k+1}w_1\underline{o(j)}w_2$ where the indicated occurrence of $\underline{o(j)}$ is the termination symbol corresponding to x_{k+1}.

Then, for any w_1', w_1'' with $w_1 = w_1' w_1''$ we have $\langle \alpha_0 \cdot pvx_{k+1} w_1' \underline{o(j)} w_1'' \rangle$ and by repeated application of condition (*) it follows that

$$\alpha_0 \cdot pv\overline{o(j)}\,\underline{o(j)}\, w_1 w_2 = \alpha_0 \cdot pv\,\overline{o(j)}\, w_1 \underline{o(j)}\, w_2.$$

We have convinced ourselves that we can replace our computation $z(k)$ by a computation $z_1(k)$ of the form $pv'\,\overline{o(j')}\,\underline{o(j')o(j)}\,v''w$, if $x_{k+1} = \underline{o(j)}$, and by a computation $z_2(k)$ of the form $pv'\,\overline{o(j')}\,\underline{o(j')}\,\overline{o(j)}\,\underline{o(j)}\,v''w$, if $x_{k+1} = \overline{o(j)}$.

Then also

$$z_1'(k) = pv'\,\underline{o(j)}\,\overline{o(j')}\,\underline{o(j')}\,v''w,$$

$$\text{resp. } z_2'(k) = pv'\,\overline{o(j)}\,\underline{o(j)}\,\overline{o(j')}\,\underline{o(j')}\,v''w,$$

are in $C(S, I)$ and because of the assumption below definition 1.10 we have $\alpha_0 \cdot z_1(k) = \alpha_0 \cdot z_1'(k)$, resp. $\alpha_0 \cdot z_2(k) = \alpha_0 \cdot z_2'(k)$. Thus we can replace $z(k)$ by $z_1'(k)$, resp. $z_2'(k)$. By an induction based on the validity of this argument we can construct a computation, that we call $z(k)$ henceforth, such that $z(k) = pvx_{k+1}u$ and, if $x_{k+1} = \overline{o(j)}$ or $x_{k+1} = \underline{o(j)}$ with $j \in N_t$ then v contains no $\overline{o(j')}$ and no $\underline{o(j')}$ with $j' \in N_t$. If v is null then we may take $z(k+1) = z(k)$. Otherwise similar interchanges as above can be used to "slide" x_{k+1} past each element of v yielding eventually the required computation $z(k+1) = (_{k+1}x)vu$.

We now show how our assumption $\Omega(x, i) \neq \Omega(y, i)$ is contradicted by the result just proven. For, if $\Omega(x, i) \neq \Omega(y, i)$, then for some $k \in N$ one of the following holds:

(i) $[\Omega(x, i)]_k$ and $[\Omega(y, i)]_k$ are both defined, but unequal;

(ii) $[\Omega(y, i)]_k$ is defined, but $[\Omega(x, i)]_k$ not;

(iii) $[\Omega(x, i)]_k$ is defined, but $[\Omega(y, i)]_k$ not.

In cases (i) and (ii) there is a suitably large n' such that $[\Omega(z(n'), i)]_k \neq [\Omega(y, i)]_k$, contradicting our assumption; case (iii) is disposed of similarly; thus condition (*) implies determinacy.

Now assume that there are computations $x' = u\sigma\pi w'$ and $x'' = u\pi\sigma w''$ in $C(S, I)$ such that $\alpha_0 \cdot u\sigma\pi \neq \alpha_0 \cdot u\pi\sigma$. Then it follows from lemma A.2 that there is an i such that $\Omega(u\sigma\pi, i)$ and $\Omega(u\pi\sigma, i)$ are of equal length, and differ in their last element; thus S is not determinate.

COROLLARY A.1: *A lossless SVAS S is determinate iff, for each $I \in I(S)$:*

(i) *if* $\langle \alpha_0 \cdot uo(i)\,\underline{o(j)} \rangle$ *and* $\langle \alpha_0 \cdot uo(j)\,\overline{o(i)} \rangle$ *then* $R(j) \cap D(i) = \emptyset$ *or* $\underline{o(j)}$ *is a repetition and,*

(ii) *if* $\langle \alpha_0 \cdot uo(i)o(j) \rangle$ *and* $\langle \alpha_0 \cdot uo(j)o(i) \rangle$ *then* $R(i) \cap R(j) = \emptyset$.

Proof: Follows directly from theorem A.1 and lemma A.2.

COROLLARY A.2: *Let S be a repetition-free, lossless SVAS. Then the following statements are equivalent:*

(1) *S is not determinate;*

(2) *for some* $I \in I(S)$ *there are* $u \in O^*$, $o(i)$, $o(j) \in \mathrm{Op}$ *with either:*

(i) $R(j) \cap D(i) \neq \emptyset$ *and* $\langle \alpha_0 \cdot u\overline{o(i)}\,\underline{o(j)} \rangle$ *and* $\langle \alpha_0 \cdot u\underline{o(j)}\,\overline{o(i)} \rangle$ or

(ii) $R(i) \cap R(j) \neq \emptyset$ *and* $\langle \alpha_0 \cdot u\underline{o(i)\,o(j)} \rangle$ *and* $\langle \alpha_0 \cdot u\underline{o(j)\,o(i)} \rangle$;

(3) *for some* $I \in I(S)$ *there are* $w \in O^*$, $o(i)$, $o(j) \in \mathrm{Op}$ *with* $(i, j) \in K$ *and* $\langle \alpha_0 \cdot w\overline{o(i)} \rangle$ *and* $\langle \alpha_0 \cdot w\overline{o(j)} \rangle$;

(4) *for some* $I \in I(S)$ *there are* $x, y \in C(S, I)$ *and* $o(i)$, $o(j)$ *such that* $(i, j) \in K$ *and the subsequence* $E_{\overline{o(i)\,o(j)}}(x)$ *obtained by extracting all occurrences of* $\overline{o(i)}$ *and* $\overline{o(j)}$ *from x differs from the subsequence* $E_{\overline{o(i)\,o(j)}}(y)$ *similarly obtained from y.*

Proof: Analogous to the proof of corollary 2.2 of [9].

APPENDIX II

In this appendix we execute algorithm 2.1 on the Strassen-Winograd algorithm for multiplication of 2×2 matrices. In order to conform with the notation in [1] we use indexed capital letters for the operations and indexed lower case letters for the domains and ranges of the operations.

The Strassen-Winograd algorithm multiplies two 2×2 matrices (a_{ij}) and (b_{ij}) into the product 2×2 matrix (c_{ij}) using seven scalar multiplications $M_1, \ldots, M_7$ and 15 scalar additions $S_1, \ldots, S_8, T_1, T_2, C_1, C_2, C_3, C_4$. We call M the execution time of one scalar multiplication and A the execution time of one scalar addition. The memory locations called $s_1, \ldots, s_8, m_1, \ldots, m_7, t_1, t_2$ store intermediate results. The domains and ranges as well as the execution times of the operations can be found in table 1. The precedence graph of the Strassen-Winograd algorithm is shown in figure 1; under each operation the corresponding value of e is displayed. Figure 2 shows a "dump" of the execution of algorithm 2.1 on the Strassen-Winograd algorithm under the (natural) assumption of $M > 3A$. All assignments of values to C, T, T', I and I' are shown as well as all calls of the procedure NEXT; also the value of $\min \{ e(i') \mid i' \in I \}$ is exhibited under "min". Figure 3 shows the computation x that algorithm 2.1 finds after the execution dumped in figure 2 as well as the values of m on each prefix of x. We have $m(x) = M + 5A$. One can similarly see that also for $A \leqq M \leqq 3A$ algorithm 2.1 finds a computation x' with $m(x') = M + 5A$.

TABLE 1

operation	domain	range	execution time
S_1	a_{21} a_{22}	s_1	A
S_2	s_1 a_{11}	s_2	A
S_3	a_{11} a_{21}	s_3	A
S_4	a_{12} s_2	s_4	A
S_5	b_{12} b_{11}	s_5	A
S_6	b_{22} s_5	s_6	A
S_7	b_{22} b_{12}	s_7	A
S_8	s_6 b_{21}	s_8	A
M_1	s_2 s_6	m_1	M
M_2	a_{11} b_{11}	m_2	M
M_3	a_{12} b_{21}	m_3	M
M_4	s_3 s_7	m_4	M
M_5	s_1 s_5	m_5	M
M_6	s_4 b_{22}	m_6	M
M_7	a_{22} s_8	m_7	M
T_1	m_1 m_2	t_1	A
T_2	t_1 m_4	t_2	A
C_1	m_2 m_3	c_{11}	A
C_2	t_1 m_5 m_6	c_{12}	2A
C_3	t_2 m_7	c_{21}	A
C_4	t_2 m_5	c_{22}	A

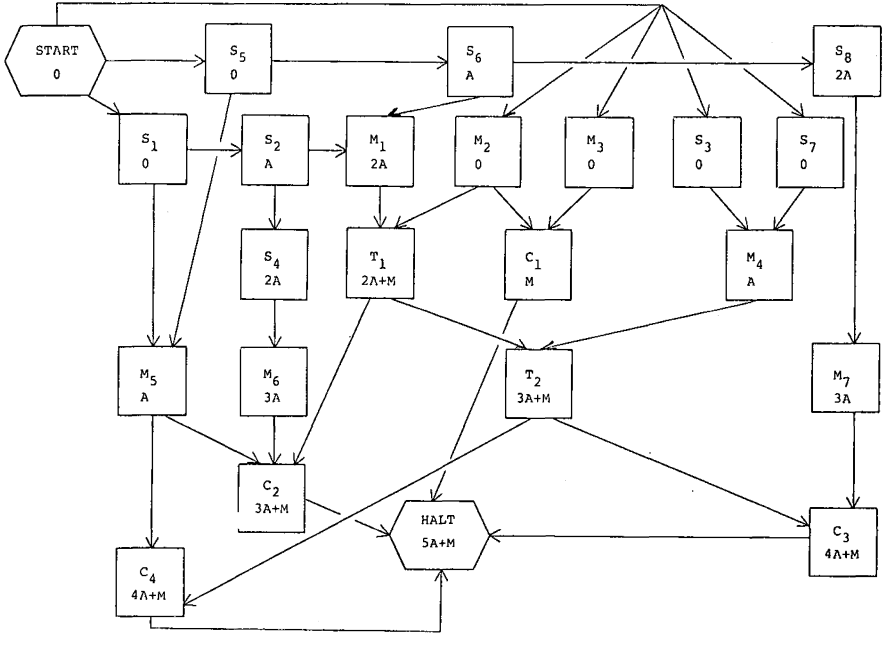

Figure 1

$T = \emptyset$

$I = \emptyset$

NEXT

$C = \{S_1, S_5, M_2, M_3, S_3, S_7\}$

$T = \{S_1, S_5, M_2, M_3, S_3, S_7\}$

$I = \{S_1, S_5, M_2, M_3, S_3, S_7\}$ min = 0

$T' = \emptyset$

NEXT

$C = \{S_2, S_6, M_5, C_1, M_4\}$

$T = \{S_1, S_5, M_2, M_3, S_3, S_7, S_2, S_6, M_5, C_1, M_4\}$

$I = \{S_1, S_5, M_2, M_3, S_3, S_7, S_2, S_6, M_5, C_1, M_4\}$ min = 0

$I' = \{S_1, S_5, M_2, M_3, S_3, S_7\}$

$I = \{S_2, S_6, M_5, C_1, M_4\}$ min = A

$T' = \{S_1, S_5, S_3, S_7\}$

$T = \{M_2, M_3, S_2, S_6, M_5, C_1, M_4\}$

NEXT

$C = \{M_1, S_8, S_4\}$

$T = \{M_2, M_3, S_2, S_6, M_5, C_1, M_4, M_1, S_8, S_4\}$

$I = \{S_2, S_6, M_5, C_1, M_4, M_1, S_8, S_4\}$ min = A

$I' = \{S_2, S_6, M_5, M_4\}$

$I = \{C_1, M_1, S_8, S_4\}$ min = 2A

$T' = \{S_2, S_6\}$

$T = \{M_2, M_3, M_4, M_5, M_1, C_1, S_4, S_8\}$

NEXT

$C = \{T_1, M_6, M_7\}$

$T = \{M_1, M_2, M_3, M_4, M_5, M_6, M_7, S_4, S_8, C_1, T_1\}$

$I = \{M_1, M_6, M_7, S_4, S_8, C_1, T_1\}$ min = 2A

$I' = \{M_1, S_4, S_8\}$

$I = \{M_6, M_7, C_1, T_1\}$ min = 3A

$T' = \{S_4, S_8\}$

$T = \{M_1, M_2, M_3, M_4, M_5, M_6, M_7, C_1, T_1\}$

NEXT

$C = \{C_2, T_2\}$

$T = \{M_1, M_2, M_3, M_4, M_5, M_6, M_7, C_1, C_2, T_1, T_2\}$

$I = \{M_6, M_7, C_1, C_2, T_1, T_2\}$ min = 3A

$I' = \{M_6, M_7\}$

$I = \{C_1, C_2, T_1, T_2\}$ min = M

$T' = \{M_2, M_3\}$

$T = \{M_1, M_4, M_5, M_6, M_7, C_1, C_2, T_1, T_2\}$

NEXT

$C = \{C_3, C_4\}$

$T = \{M_1, M_4, M_5, M_6, M_7, C_1, C_2, C_3, C_4, T_1, T_2\}$

$I = \{C_1, C_2, C_3, C_4, T_1, T_2\}$ min = M

$I' = \{C_1\}$

$I = \{T_1, T_2, C_2, C_3, C_4\}$ min = 2A+M

$T' = \{M_1, M_4, M_5, C_1\}$

$T = \{M_6, M_7, C_2, C_3, C_4, T_1, T_2\}$

NEXT

$C = \{HALT\}$

$T = \{M_6, M_7, C_2, C_3, C_4, T_1, T_2, HALT\}$

$I = \{T_1, T_2, C_2, C_3, C_4, HALT\}$ min = 2A+M

$I' = \{T_1\}$

$I = \{T_2, C_2, C_3, C_4, HALT\}$ min = 3A+M

$T' = \{M_6, M_7, T_1\}$

$T = \{C_2, C_3, C_4, T_2, HALT\}$

NEXT

$C = \emptyset$

$I' = \{C_2, T_2\}$

$I = \{C_3, C_4, HALT\}$ min = 4A+M

$T' = \{T_2\}$

$T = \{C_2, C_3, C_4, HALT\}$

NEXT

$C = \emptyset$

$I' = \{C_3, C_4\}$

$I = \{HALT\}$ min = 5A+M

$T' = \{C_2, C_3, C_4\}$

$T = \{HALT\}$

NEXT

$C = \emptyset$

$I' = \{HALT\}$

$I = \emptyset$

Figure 2

x =	$\bar{S}_1$	$\bar{S}_5$	$\bar{M}_2$	$\bar{M}_3$	$\bar{S}_3$	$\bar{S}_7$	$\underline{S}_1$	$\underline{S}_5$	$\underline{S}_3$	$\underline{S}_7$	$\bar{M}_4$	$\bar{M}_5$
m	0	..	..	..	..	0	A	..	..	..	..	..

	$\bar{S}_2$	$\bar{S}_6$	$\underline{S}_2$	$\underline{S}_6$	$\bar{S}_4$	$\bar{S}_8$	$\bar{M}_1$	$\underline{S}_4$	$\underline{S}_8$	$\bar{M}_6$	$\bar{M}_7$	$\underline{M}_2$
m	..	A	2A	..	..	..	2A	3A	..	..	3A	M

	$\underline{M}_3$	$\bar{C}_1$	$\underline{M}_1$	$\underline{M}_4$	$\underline{M}_5$	$\underline{C}_1$	$\bar{T}_1$	$\underline{T}_1$	$\underline{M}_6$	$\underline{M}_7$	$\bar{T}_2$	$\bar{C}_2$
m	..	M	M+2A	..	..	..	M+2A	M+3A	..	..	..	M+3A

	$\underline{T}_2$	$\bar{C}_3$	$\bar{C}_4$	$\underline{C}_2$	$\underline{C}_3$	$\underline{C}_4$
m	M+4A	..	M+4A	M+5A	..	M+5A

m(x) = M+5A.

Figure 3

y =	$\bar{S}_1$	$\bar{S}_3$	$\bar{S}_5$	$\bar{S}_7$	$\underline{S}_1$	$\underline{S}_3$	$\underline{S}_5$	$\underline{S}_7$	$\bar{S}_2$	$\bar{S}_6$	$\underline{S}_2$	$\underline{S}_6$
m	0	..	..	0	A	..	..	..	..	A	2A	..

	$\bar{S}_4$	$\bar{S}_8$	$\underline{S}_4$	$\underline{S}_8$	$\bar{M}_1$	$\bar{M}_2$	$\bar{M}_3$	$\bar{M}_4$	$\bar{M}_5$	$\bar{M}_6$	$\bar{M}_7$	$\underline{M}_1$
m	..	2A	3A	..	..	..	..	..	..	..	3A	M+3A

	$\underline{M}_2$	$\underline{M}_3$	$\underline{M}_4$	$\underline{M}_5$	$\underline{M}_6$	$\underline{M}_7$	$\bar{T}_1$	$\underline{T}_1$	$\bar{T}_2$	$\underline{T}_2$	$\bar{C}_1$	$\bar{C}_2$
m	..	..	..	..	..	..	M+3A	M+4A	M+4A	M+5A	..	..

	$\bar{C}_3$	$\bar{C}_4$	$\underline{C}_1$	$\underline{C}_2$	$\underline{C}_3$	$\underline{C}_4$
m	..	M+5A	M+6A	M+7A	..	M+7A

m(y) = M+7A.

Figure 4

Finally figure 4 exhibits a parallel computation y that one would get from the sequential execution of the Strassen-Winograd algorithm using only the "obvious" possibilities of parallelization effecting a minimization of m; we have $m(y) = M + 7A$.