A BOUNDARY BETWEEN DECIDABILITY AND UNDECIDABILITY FOR PARALLEL PROGRAM SCHEMATA (Extended Abstract)

Raymond E. Miller
Mathematical Sciences Department
IBM Thomas J. Watson Research Center
Yorktown Heights, New York

ABSTRACT

Some theorems showing undecidability for computational commutativity, boundedness, termination and determinacy of parallel program schemata are given. These results are then compared with contrasting decidability results in [1] showing that the deletion of the hypothesis of repetition-freeness from the decidability theorems produces undecidability.

I. INTRODUCTION

In attempts to better understand the structure of computer programs and to circumvent the well known undecidability results concerning, for example, the termination and equivalence of programs, people have turned to modeling certain restricted aspects of programs for which some more positive (decidability) results can be obtained [1-5]. The determination and formulation of important properties of program structure and behavior are of interest in themselves. In addition however, results that establish boundaries between when such inherent properties are decidable or undecidable provide a fuller understanding of what modifications and simplifications of programs can be carried out in an algorithmic fashion.

In this paper we extend some undecidability results of [5] to detecting computational commutativity, boundedness, termination and determinacy. In particular, the unsolvability of determinacy holds for finite-state schemata (Theorem 4). This answers negatively a problem posed in [1] concerning the existence of an effective test for determinacy for the class of finite state schemata. In Section V we compare these results with the decidability results of [1], and show that the deletion of the single hypothesis of repetition-freeness provides a boundary between decidability and undecidability for these properties of parallel program schemata.

II. PRELIMINARIES

To make this paper fairly self-contained we review some of the basic definitions of parallel program schemata [1].

A parallel program schema $\mathcal{S} = (M, A, \mathcal{T})$ consists of: a set M of memory locations; a finite set A of operations: we associate with each $a \in A$ a positive integer $K(a)$ called the number of outcomes of a, and sets $D(a) \subseteq M$ and $R(a) \subseteq M$ called the domain locations and range locations of a respectively; and a transition system control $\mathcal{T} = (Q, q_0, \Sigma, \tau)$ where Q is a set of states, q_0 is a designated initial state, $\Sigma = \Sigma_i \cup \Sigma_t$ is the alphabet where $\Sigma_i = \bigcup_{a \in A} \{\bar{a}\}$ is the set of initiation symbols and $\Sigma_t = \bigcup_{a \in A} \{a_1, \ldots, a_{K(a)}\}$ is the set of termination symbols, and τ is a partial transition function from $Q \times \Sigma$ to Q which is total on $Q \times \Sigma_t$.

An interpretation $\mathcal{I}$ of a schema $\mathcal{S}$ is given by a function C which associates a set of values $C(i)$ with each $i \in M$, an initial memory contents $c_0 \in \times_{i \in M} C(i)$, and for each $a \in A$ two functions $F_a : \times_{i \in D(a)} C(i) \to \times_{i \in R(a)} C(i)$ and $G_a : \times_{i \in D(a)} C(i) \to \{a_1, \ldots, a_{K(a)}\}$. For a performance of a, F_a determines the results to be stored in locations $R(a)$ and G_a determines the conditional branch to be taken.

A finite or infinite word z over Σ is called an $\mathcal{I}$-computation for $\mathcal{S}$ if for c_0, F_a and G_a defined by $\mathcal{I}$: (1) every prefix $y\sigma$ of z with $\sigma \in \Sigma$ satisfies the constraints that $\tau(q_0, y\sigma)$ is defined, and if σ is a termination symbol for $a \in A$ then the number of initiation symbols $\bar{a}$ in y is greater than the number of termination symbols in y for operation a; (2) if z is finite then for all $\sigma \in \Sigma$ Condition (1) is not satisfied for $z\sigma$; (3) if x is a prefix of z and $\sigma \in \Sigma$ with the property that for every y such that xy is a prefix of z it follows that $xy\sigma$ satisfies (1), then for some y', $xy'\sigma$ is a prefix of z; (4) if $x\sigma$ is a prefix of z and $\sigma \in \Sigma_t$ where σ is the ith termination symbol of operation a in $x\sigma$ then G_a evaluated after the ith $\bar{a}$ in x equals σ.

An $\mathcal{I}$-computation z thereby represents a sequence of initiations and terminations of operations which is consistent with the schema control $\mathcal{T}$ and the outcome function G_a. The memory locations are read and changed by the sequence of initiations and terminations. Upon the initiation of an operation a the values in locations $D(a)$ are read. These values are then used both to compute new values for locations in the set $R(a)$ in accord with the function F_a, and to determine the outcome of a defined by G_a for this performance of operation a. Upon termination of the operation a the values computed by F_a are stored

in locations R(a). In this way an $\mathcal{J}$-computation z defines a sequence of contents for each cell $i \in M$ and we denote this sequence by $\Omega_i(z)$. A more detailed definition of computations and the resulting sequences of memory values is given in [1] but this description should suffice for our current purposes.

A schema is called determinate if and only if for each pair of $\mathcal{J}$-computations y and z, $\Omega_i(y) = \Omega_i(z)$ for each $i \in M$. Two schemata $\mathcal{S} = (M, A, \mathcal{T})$ and $\mathcal{S}' = (M, A, \mathcal{T}')$ are called equivalent if for each $i \in M$ and each interpretation $\mathcal{J}$:

$\{\Omega_i(y) \mid y$ is an $\mathcal{J}$-computation for $\mathcal{S}\} = \{\Omega_i(z) \mid z$ is an $\mathcal{J}$-computation for $\mathcal{S}'\}$.

A schema $\mathcal{S}$ is called bounded if there is a constant K such that for every interpretation $\mathcal{J}$ any prefix x of any $\mathcal{J}$-computation has a number of initiation symbols which is no more than K greater than the number of termination symbols in x. If K can be taken as 1 then the schema is said to be serial. Suppose for schema $\mathcal{S}$, q is an arbitrary state in Q, and σ and π are arbitrary distinct elements of Σ. $\mathcal{S}$ is persistent if whenever $\tau(q,\sigma)$ and $\tau(q,\pi)$ are defined then $\tau(q,\sigma\pi)$ and $\tau(q,\pi\sigma)$ are also defined. $\mathcal{S}$ is called commutative if whenever $\tau(q,\pi\sigma)$ and $\tau(q,\sigma\pi)$ are defined then $\tau(q,\pi\sigma) = \tau(q,\sigma\pi)$. $\mathcal{S}$ is computationally commutative if whenever for some given interpretation $\mathcal{J}$, $x\pi\sigma$ and $x\sigma\pi$ are prefixes of $\mathcal{J}$-computations then $\tau(q_0, x\pi\sigma) = \tau(q_0, x\sigma\pi)$. If π and σ are both initiation symbols $\mathcal{S}$ is permutable if whenever $\tau(q,\sigma\pi)$ is defined then $\tau(q,\pi)$ is also defined. $\mathcal{S}$ is called lossless if, for every $a \in A$, $R(a) \neq \emptyset$. $\mathcal{S}$ is repetition-free if whenever an $\mathcal{J}$-computation contains two initiation symbols of the same operation such as $v\,\bar{a}\,w\,\bar{a}\,x$ then w contains a termination symbol of an operation c for which $R(c) \cap D(a) \neq \emptyset$. Finally, an operation $a \in A$ is said to be terminating if $\bar{a}$ occurs only a finite number of times in each computation of $\mathcal{S}$.

A counter schema is a schema whose control $\mathcal{T}$ is specified by: a nonnegative integer k (the number of counters), a finite set S with a distinguished element s_0; a vector $\Pi \in N^k$, where N denotes the nonnegative integers; a function v from the alphabet Σ into N^k such that $\sigma \in \Sigma_t$ implies that $v(\sigma) \geq 0$; and a partial function $\theta : S \times \Sigma \rightarrow S$ which is total on $S \times \Sigma_t$. Now for the control $\mathcal{T} = (Q, q_0, \Sigma, \tau)$ of the counter schema the state set $Q = S \times N^k$, $q_0 = (s_0, \Pi)$, and $\tau((s,x),\sigma)$ is defined if $\theta(s,\sigma)$ is defined and $x + v(\sigma) \geq 0$; when defined, $\tau((s,x),\sigma) = (\theta(s,\sigma), x + v(\sigma))$.

III. THE BASIC CONSTRUCTION

The elements of the constructions we use to prove undecidability are taken from [1] along with the "reset" idea of [5]. In those papers, as well as here, undecidability is proven by using the Post correspondence problem. The form of the Post correspondence problem we use can be stated as follows: Given two n-tuples $X = x_1, x_2, \ldots, x_n$ and $Y = y_1, y_2, \ldots, y_n$ of words over the alphabet $\{b_1, b_2\}$, to decide whether there exists a sequence of indices $i_1, i_2, \ldots, i_p$ such that $x_{i_1} x_{i_2} \cdots x_{i_p} = y_{i_1} y_{i_2} \cdots y_{i_p}$. This problem is denoted as P(X, Y). We show that the properties we wish to study for schemata are decidable only if this class of Post correspondence problems is decidable. Since undecidability holds for this class of problems, this is sufficient to prove undecidability for the schemata properties.

For any particular Post correspondence problem P(X, Y) we construct an $\mathcal{S}(X)$ and $\mathcal{S}(Y)$ similar to that in [1]. For $\mathcal{S}(X)$ and $\mathcal{S}(Y)$: $M = \{1,2\}$, $A = \{a,b\}$, $D(a) = R(a) = \{1\}$, $D(b) = R(b) = \{2\}$ $K(a) = K(b) = 3$. Since neither operation affects the domain location of the other, the sequence of outcomes of a and b depend only on the interpretation and not on how the performance of a and b are interspersed. $\mathcal{S}(X)$ and $\mathcal{S}(Y)$ are constructed in an identical manner so we describe the construction of $\mathcal{S}(X)$ only. We say that an interpretation $\mathcal{J}$ is consistent with $(X; i_1, \ldots, i_p)$ if and only if:

(i) if a could be executed repeatedly, beginning with the control in state q_0 and the initial assigned contents of memory location 1, the sequence of outcomes would have as a prefix:

$$a_1^{i_1-1} \quad a_2 a_1^{i_2-1} \quad a_2 \cdots a_1^{i_p-1} \quad a_2 a_3$$

and

(ii) if b could be executed repeatedly, beginning with state q_0 and the initial assigned contents of memory location 2, the sequence of outcomes would have the prefix:

$$x_{i_1} x_{i_2} \cdots x_{i_p} b_3.$$

Thus, $\mathcal{S}(X)$ is designed so that under a consistent interpretation the outcomes of a determine a sequence of indices and the outcomes of b determine the word generated from X by this sequence of indices. The actual computation for $\mathcal{S}(X)$ under a consistent interpretation would have performances of a and b interspersed so that the sequence of outcomes would be

$$a_1^{i_1-1} \quad a_2 x_{i_1} a_1^{i_2-1} \quad a_2 x_{i_2} \cdots a_1^{i_p-1} \quad a_2 x_{i_p} a_3 b_3.$$

A control for $\mathcal{S}(X)$ to accomplish this is sketched in Figure 1. For termination symbol transitions not shown in Figure 1 the $\mathcal{S}(X)$ transitions are all assumed to go to a sink state construction which is serial and nonterminating.

From the construction of $\mathcal{S}(X)$ it is readily seen that for any pair $(X;i_1,i_2,\ldots,i_p)$ and interpretation $\mathcal{I}$ state q_e is reached in $\mathcal{S}(X)$ if and only if $\mathcal{I}$ is consistent with $(X;i_1,i_2,\ldots,i_p)$. If $\mathcal{I}$ is not consistent with $(X;i_1,i_2,\ldots,i_p)$ then each $\mathcal{I}$-computation reaches the "sink" and is infinite in length.

Now consider schema $\mathcal{S}(XY)$ depicted Figure 2.

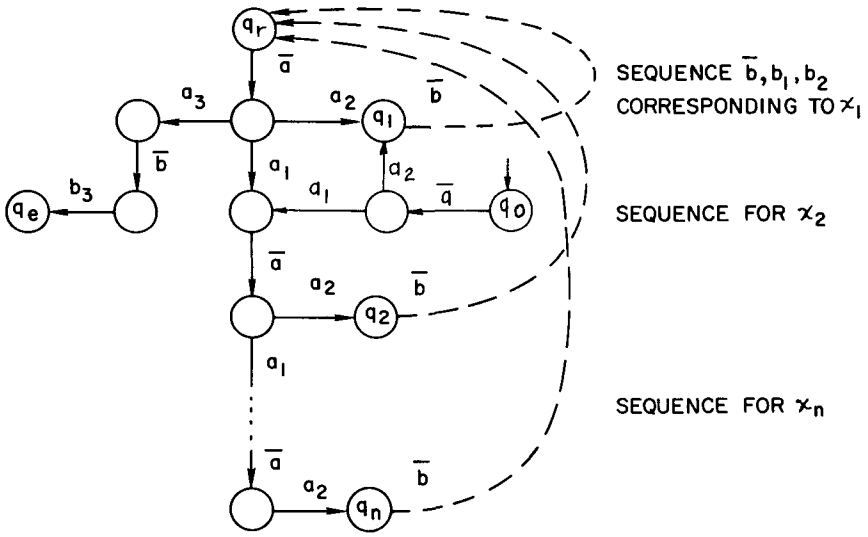

Figure 1: Sketch of $\mathcal{S}(X)$ Control

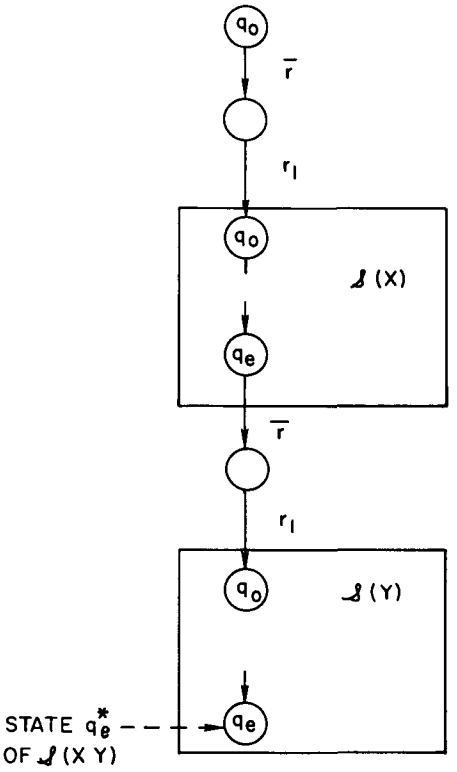

Figure 2: Schema $\mathcal{S}(X\,Y)$ with reset operation r

In schema $\mathcal{S}(X\,Y)$ we let $M = \{0,1,2\}$ $D(r) = 0$, $R(r) = (1,2)$ and operations a and b be defined as before for $\mathcal{S}(X)$ and $\mathcal{S}(Y)$. For convenience we denote state q_e of $\mathcal{S}(Y)$ in $\mathcal{S}(X\,Y)$ as q_e^*. Note that the first performance of operation r, preceding $\mathcal{S}(X)$, initializes locations 1 and 2. The interpretation is consistent with $(X;i_1,i_2,\ldots,i_p)$ for some $(i_1,i_2,\ldots,i_p)$ if and only if the computation reaches state q_e in the $\mathcal{S}(X)$ part of $\mathcal{S}(X\,Y)$. In this event operation r is performed a second time, and state q_0 of $\mathcal{S}(Y)$ is entered. The second performance of r must reset locations 1 and 2 to the same values that they had upon entering $\mathcal{S}(X)$ (since F_r is single valued, and location 0 is never changed). Thus, the sequence of outcomes for a in $\mathcal{S}(Y)$ must have the same prefix as the sequence that occurred in $\mathcal{S}(X)$;

in particular the outcomes must have the prefix $a_1^{i_1-1}\,a_2\,a_1^{i_2-1}\,a_2\ldots a_1^{i_p-1}\,a_2a_3$. Now, the interpretation is also consistent with Y if and only if state q_e^* of $\mathcal{S}(X\,Y)$ is reached. Thus q_e^* is reached in some computation if and only if there is a solution to the Post correspondence problem P(X, Y). Thus the accessibility of states, in particular state q_e^*, is undecidable for this type of schema.

From the construction it can be shown that $\mathcal{S}(XY)$ is a finite state, one-valued, serial, determinate, permutable, persistent, computationally commutative, lossless, counter schema.

IV. THE UNDECIDABILITY THEOREMS

Using $\mathcal{S}(X\,Y)$ and minor variations the following new undecidability results can be obtained.

Theorem 1: It is undecidable whether a given finite state determinate, permutable, persistent, lossless, counter schema is computationally commutative or serial.

Theorem 2: It is undecidable whether a given finite state, determinate, permutable, persistent, computationally commutative, lossless, counter schema is bounded.

Theorem 3: It is undecidable for a finite state, serial, determinate, permutable, persistent, computationally commutative, lossless, counter schema whether a given operation $a \epsilon A$ is terminating.

It is interesting to note that one sees directly from $\mathcal{S}(XY)$ that the question of the existence of a finite computation is undecidable. From Theorem 3 we obtain a different result, namely that the question of the existence of an infinite computation is undecidable.

Theorem 4: It is undecidable whether a finite state, permutable, persistent, computationally commutative, lossless, counter schema is determinate.

It can also be shown that in Theorem 4 the terms "persistent" and "permutable" can be removed and replaced by the term "serial".

By a simple addition of an operation which is performed exactly once only when q_e^* of $\mathcal{S}(XY)$ is reached we can obtain the related result:

Theorem 5: It is undecidable for a finite state, serial, determinate, permutable, persistent, computationally commutative, lossless, counter schema whether, for a given operation c, any computation exists containing $\overline{c}$.

V. THE BOUNDARY

In [1] a number of results were given showing the decidability of commutativity, boundedness, termination, and determinacy for repetition-free counter schemata. The results of the previous section yield undecidability results for these properties. By comparing these results we show that deletion of repetition-freeness from the hypothesis of the decidability theorems changes each problem to an undecidable one. Thus, in a sense, we are "close" to the borderline between decidability and undecidability.

We proceed with this comparison.

Theorem (4.5 of [1]): It is decidable whether a given repetition-free counter schema $\mathcal{S}$ is commutative.

By inspecting the proof of this theorem it becomes evident that the result is also true if "computationally commutative" replaces commutative.

Corollary (of Theorem 1): It is undecidable whether a given counter schema is computationally commutative.

Theorem (4.6 of [1]): It is decidable whether a given repetition-free counter schema is bounded.

Corollary (of Theorem 2): It is undecidable whether a given counter schema is bounded.

Theorem (4.7 of [1]): It is decidable of a repetition-free counter schema $\mathcal{S}$ whether a given operation $a \epsilon A$ is terminating.

Corollary (of Theorem 3): It is undecidable of a counter schema $\mathcal{S}$ whether a given operation $a \epsilon A$ is terminating.

Theorem (4.9 of [1]): It is decidable whether a repetition-free, lossless, persistent, commutative counter schema is determinate.

In this theorem the term commutative can also be replaced by "computationally commutative" since determinacy is a property that is concerned only with sequences that can arise from computations.

Corollary (of Theorem 4): It is undecidable whether a lossless, persistent, computationally commutative, counter schema is determinate.

From these theorem-corollary pairs it is evident that the removal of the single hypothesis of repetition-freeness changes the problem in question from a decidable problem to an undecidable one. Since only the one hypothesis is removed, these results are as tight as can be expected.

It is interesting to note that the schema $\mathcal{S}(XY)$, which is the basis for the undecidability results, is itself very reliant on the repetitive character of the "reset" operation. In particular, operation r is the only repetitive operation of $\mathcal{S}(XY)$ and is performed at most twice in any computation. Thus $\mathcal{S}(XY)$ is in some sense minimally repetitive since only one operation can be repetitive and this operation can be repeated only once.

The corollaries of Theorems 1 through 4 were obtained simply by deleting some of the schema constraints of the theorems. These constraints, or any combination of them, could be added back as hypotheses in the respective corollaries. Of course, the constraints could also be added into the hypotheses of the decidability theorems for each of the theorem-corollary pairs since adding further constraints to the hypotheses of the decidability theorems could only tend to simplify the problem further. This thereby gives

a family of comparable theorem-corollary pairs with whatever combination of constraining hypotheses consistent with Theorems 1 through 4 are desired.

Since it is decidable whether a given counter schema is repetition-free, (Theorem 4.4 of [1]) we can now see rather clearly the importance of repetition-freeness in schemata. A natural question for further study arises from these results as to whether similar results can be obtained by deletion of one or more of the other constraining hypotheses in the decidable theorems. For example, does undecidability result if either persistence or computational commutativity are deleted from the determinacy theorem? Since $\mathcal{J}(X\,Y)$ is repetitive it is clear that a different basic construction would be required to answer these questions.

Acknowledgment

The author is grateful to Dr. Arnold L. Rosenberg for his careful reading and detailed comments on an earlier version of this paper.

References

[1] Richard M. Karp and Raymond E. Miller, "Parallel Program Schemata", Journal of Computer and System Sciences, Vol. 3, No. 2, May 1969, pp. 147-195.

[2] I. I. Ianov, "The Logical Schemes of Algorithms", Problems of Cybernetics, Vol. 1, 1958, pp. 75-127.

[3] J. D. Rutledge, "On Ianov's Program Schemata", Journal of the Association for Computing Machinery, Vol. 11, January 1964, pp. 1-9.

[4] D. C. Luckham, D. M. R. Park, and M. S. Paterson, "On Formalized Computer Programs", Journal of Computer and System Sciences, Vol. 4, No. 3, June 1970, pp. 220-249.

[5] V. E. Itkin and Z. Zwinogrodzki, "On the Program Schemata Equivalence", report of the Computing Center of the Siberian Branch of the USSR Academy of Sciences, Novosibirsk 90, USSR. 1970.