Synchronous Schemes and Their Decision Problems
(Extended Abstract)

Zohar Manna
Stanford University and
Weizmann Institute of Science
and
Amir Pnueli
Tel Aviv University

Abstract.

A class of schemes called synchronous schemes is defined. A synchronous scheme can have several variables, but all the active ones are required to keep a synchronized rate of computation as measured by the height of their respective Herbrand values. A "reset" statement, which causes all the variables to restart a new computation, is admitted. It is shown that equivalence, convergence, and other properties are decidable for schemes in this class. The class of synchronous schemes contains, as special cases, the known decidable classes of Ianov schemes, one-variable schemes with resets, and progressive schemes.

Introduction.

As is well-known, equivalence, convergence, and other properties of Ianov schemes ([I],[R]) and, more generally, of one-variable schemes ([CM],[C]) are decidable. The deep reason for this is not the restriction to a single variable but the fact that the computation progresses in a uniform manner.

The only workable tool that has been developed to date for analyzing general schemes is that of "unwinding and annotating" ([P],[M],[G]): The loops of the scheme are unwound continuously. During this process, the interrelations between variables that are implied by the assignments, and the truth values of predicates that are implied by the outcomes of tests, are recorded. The entrance of each generated statement box is annotated with the accumulated information known at this point. The utility of this attached information is that whenever we encounter a test whose result can be deduced from the information currently available, we can forego the test and just pursue the branch corresponding to the known truth value. If the process terminates, the resulting scheme is "free", i.e., it has the desirable property that every path in the scheme graph is realizable by some computation.

The convergence and divergence problems of free schemes are decidable, bot for all interpretations or for some interpretation. Indeed, these problems are decided by syntactical inspection: A free scheme converges for some interpretation if it contains a 'halt' statement, and it converges for all interpretations if it does not contain loops in its graph or explicit 'loop' statements.

Moreover, if a scheme is known to be free, and is fully annotated in the sense that all known interrelations between the values of program variables are recorded at any point, we can similarly resolve the following "inner equivalence" problem: "For a scheme with output variables z_1 and z_2, is it true that, for any interpretation and any computation, $z_1 = z_2$ when the computation halts?" This property holds for free and fully annotated schemes if and only if $z_1 = z_2$ is implied by the information attached to each exit box in the scheme.

This approach does not provide a decision procedure for general schemes, because the amount

---

This is an extended abstract of a forthcoming technical report of the Computer Science Dept., Stanford University, Stanford, CA, 94305.

This research was supported in part by the National Science Foundation under Grant MCS76-83655 and by the Office of Naval Research under Contract N00014-76-C-0687.

Permission to make digital or hard copies of part or all of this work or personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
© 1980 ACM 0-89791-011-7…$5.00

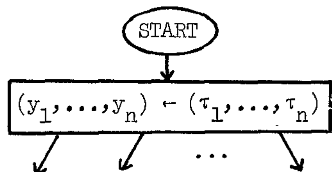
of accumulated information is unbounded, and therefore the process of unwinding and annotating a general scheme to form an equivalent free scheme may not terminate. Thus, the only solvable cases are families of schemes which are restricted in a way guaranteeing that the amount of accumulated information is bounded. Such boundedness can be obtained if we are allowed to discard already gathered information, being assured that the discarded information would never be called for again.

In this work, we define and study a class of schemes, called synchronous schemes, which is a generalization of Ianov schemes as well as of progressive schemes ([P],[LPP]). Synchronous schemes allow for many variables, but require that the values of these variables during a Herbrand computation are kept synchronized, in the sense that the differences between their heights are bounded. This enables us to discard information concerning values of height lower than the current values, and hence suggests that finite annotation will be applicable.

Terminology and Definitions.

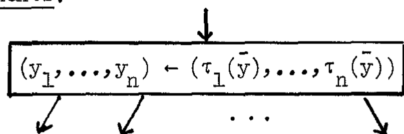
The general class of schemes that we consider are represented by flowcharts with the following types of basic statements (boxes):

Initialization:

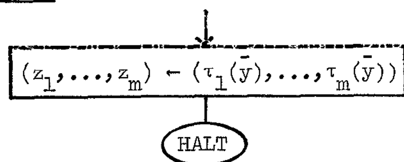


where each τ_i is an expression containing only constant symbols

Assignments:



Termination:



where each $\tau_i(\bar{y})$ is an expression that may contain constant symbols as well as the variables $\bar{y} = (y_1,\ldots,y_n)$.

These boxes are interconnected by edges which are labeled by conditions. A condition is a boolean combination of atomic formulas of the form $p(\tau_1(\bar{y}),\ldots,\tau_j(\bar{y}))$. For example, $p(f(y_1,a),g(y_2)) \wedge \sim q(g(y_1),f(b,y_2))$ is a condition.

We impose a determinism stipulation: If the edges leaving a given box are respectively labeled by the conditions $C_1,\ldots,C_p$, then these conditions must be

(a) Exclusive -- $i \neq j$ implies $\sim(C_i \wedge C_j)$, i.e., no two conditions can coexist.

(b) Exhaustive -- $C_1 \vee C_2 \vee \ldots \vee C_p = \text{true}$, i.e., at least one condition must hold.

(a) and (b) together imply that exactly one exit condition must always hold.

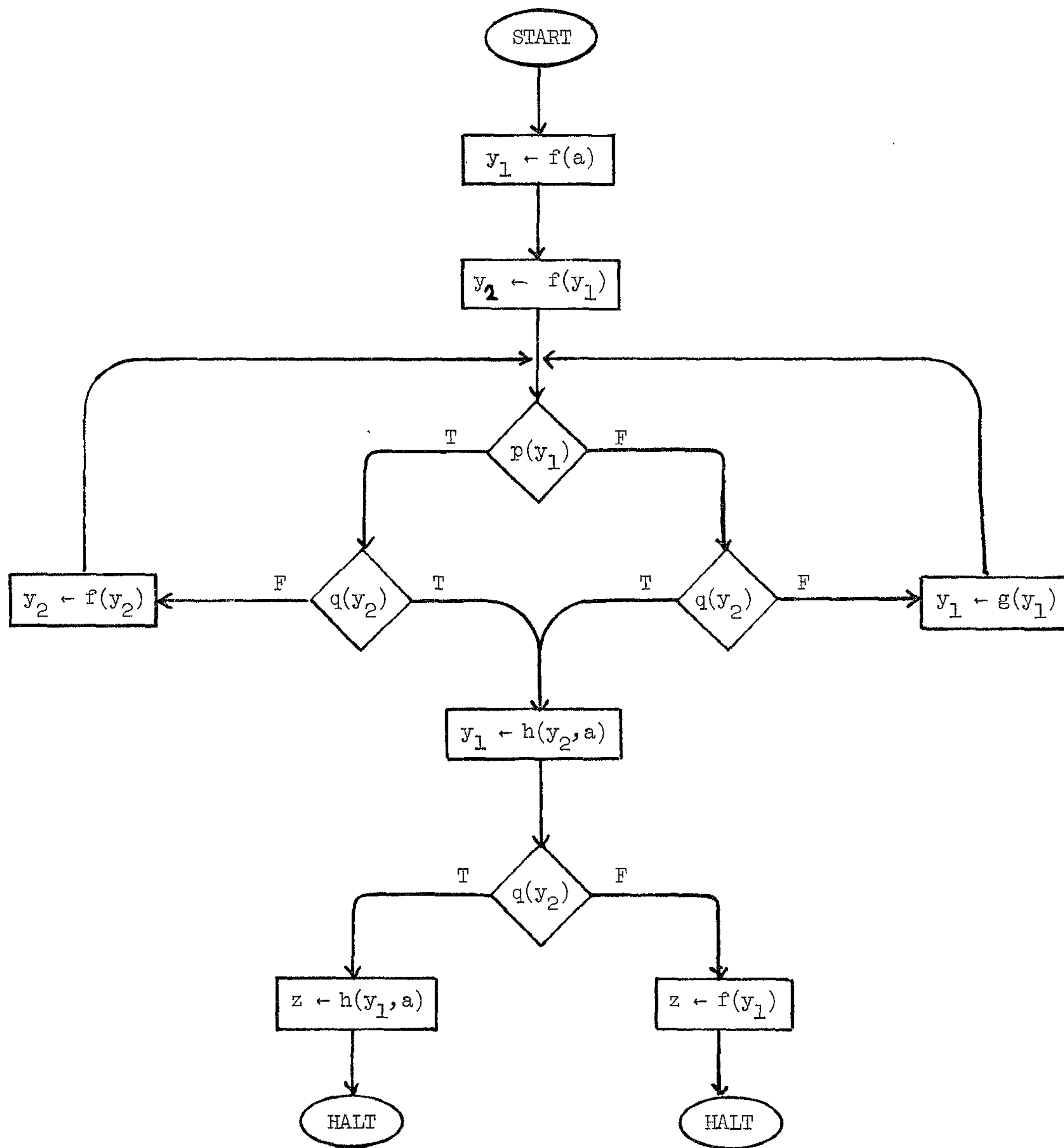
On the following page is an example of a scheme (Scheme S_1) in a conventional form and its representation (Scheme S_2) in our "transition graph" style.

A scheme is said to be simple (non-nested) if all its expressions τ are of the form $f(u_1,\ldots,u_r)$ and all its tests are literals of the form $p(u_1,\ldots,u_r)$ or $\sim p(u_1,\ldots,u_r)$, where each u_i is a variable or an individual constant.

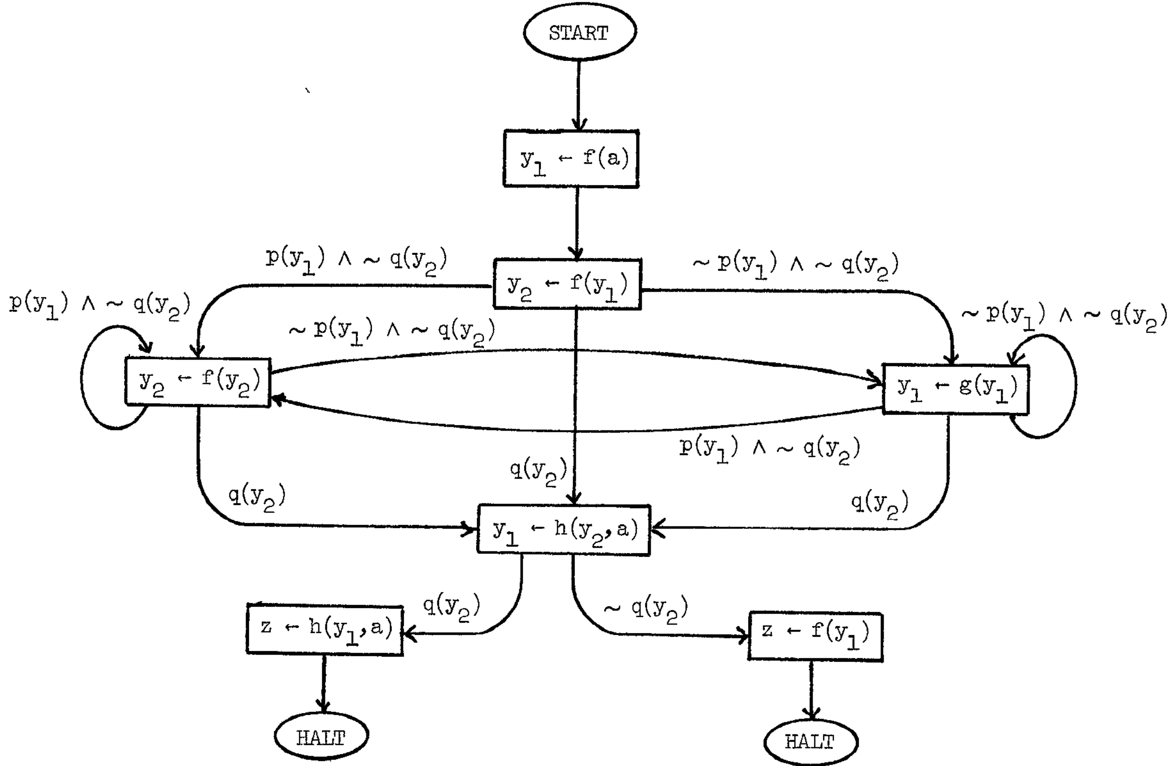
By the results in [LPP] it is sufficient to consider the behavior of schemes under the class of all Herbrand interpretations. In the sequel we consider only Herbrand domains constructed from the individual constants and function symbols of the given schemes.

For a Herbrand term t , we define its height, $|t|$, as the maximal nesting depth of the function symbols in t . Thus, for example, $|a| = 0$, $|f(a)| = |g(a,b)| = 1$, $|h(a,g(a,b))| = 2$, etc.

Consider a Herbrand computation of a scheme S . At any stage $s = 0,1,\ldots$ in the computation, each program variable y_i holds some Herbrand value, which we denote by $y_i(s)$. Let $h_i(s) = |y_i(s)|$ be the height of this value. Define also $M(s) = \max\{h_1(s),\ldots,h_n(s)\}$, i.e.,



S_1 -- A scheme in conventional form.



S_2 -- Transformation graph form of S_1 .

the maximum of the heights of the variables $y_1,\ldots,y_n$ at state s . Let $A(s)$ denote the cumulative maximum of $M(j)$ for $j = 0,1,\ldots$ up to the current state s , i.e., $A(s) = \max\{M(j) \mid 0 \leq j \leq s\}$.

A scheme is called k-synchronous for an integer $k \geq 0$, if, in every Herbrand computation, after each assignment step of the form $(y_{j_1},\ldots,y_{j_r}) \leftarrow (\tau_1,\ldots,\tau_r)$, going from state s to state $s+1$, the heights of the recently assigned values satisfy either

(a) $h_{j_1}(s+1),\ldots,h_{j_r}(s+1) \geq M(s)-k$,

or

(b) $M(s+1) \leq k$.

That is, either all the newly assigned values have height not less than k below the previous maximum, or all variables are reset to heights not exceeding k . A step of the type (b) is called a reset step.

A scheme is called synchronous if it is k-synchronous for some $k \geq 0$.

A somewhat different approach is to define a scheme to be k-monotonic if after every assignment step s assigning values to $y_{j_1},\ldots,y_{j_r}$ we have that

(a) $h_{j_1}(s+1),\ldots,h_{j_r}(s+1) \geq \tilde{A}(s)-k$,

or

(b) $M(s+1) \leq k$ as above.

Here, $\tilde{A}(s)$ is the cumulative maximum since the last reset. Thus, we relate the heights of recently computed values to the cumulative maximum $\tilde{A}(s)$ rather than to the local maximum $M(s)$.

A scheme is called monotonic if it is k-monotonic for some $k \geq 0$.

These two concepts are closely related:

Claim: A scheme is monotonic iff it is synchronous.

Obviously if a scheme is k-monotonic it is also k-synchronous. This is a direct consequence of the fact that $\tilde{A}(s) \geq M(s)$ since it is a cumulative maximum. On the other hand it can be shown that a k-synchronous scheme is always $n \cdot (k+\ell)$ -monotonic. Here n is the number of program variables and ℓ is the maximum height of any of the expressions τ appearing in the scheme.

Consider the implications of a monotonic computation. At any state of the computation some of the variables are "active" in the sense that they hold values of heights k-close to the cumulative maximum. After a finite amount of computation, unless we enter into an endless loop, the maximum must rise and some of the variables rise with it. All active computations must occur within a distance k of the maximum. Some other variables may drop behind and become "dead". A dead variable may participate in an assigned expression provided there is some live variable together with it in the same expression. A dead variable may be revitalized by assinging to it a value of height k-close to the maximum. Alternately, in a reset, all variables are reinitialized to values of height $\leq k$.

Main Results.

The class of synchronous schemes contains, as special cases:

* The class of Ianov schemes with resets.
* The more general class of one-variable schemes with resets.
* The class of progressive schemes.

These are the main classes for which equivalence is known to be decidable.

We derive the following results for synchronous schemes:

1. Every synchronous scheme S can be effectively transformed into an equivalent scheme S' which is simple, free and synchronous.

This transformation is carried out by an algorithm which will be discussed later. The algorithm checks that the scheme under transformation is actually k-synchronous and reports failure otherwise.

Hence we have:

2. For a given $k \geq 0$, it is decidable whether a scheme is k-synchronous.

However:

3. It is undecidable (though partially decidable due to (2)) whether a scheme is synchronous, that is, whether there exists a $k \geq 0$ such that the scheme is k-synchronous.

The following property enables us to formulate algorithms concerning synchronous schemes within the schemata framework without reduction to automata.

4. The class of synchronous schemes is closed under "cross product", i.e., if S_1 and S_2 are synchronous, so is $S_1 \times S_2$, properly defined.

As a consequence of the transformation (1) we have:

5. Convergence and divergence problems, both for some interpretation and for all interpretations, are decidable for synchronous schemes.

6. In comparing two synchronous schemes S_1 and S_2 , the inclusion problem $S_1 \sqsubseteq S_2$ and the equivalence problem $S_1 \approx S_2$ are both decidable.

Outline of the Algorithm.

The main technique used in establishing these results is the algorithm for transforming a synchronous scheme S into a simple, free, synchronous scheme S' as described in (1) above. Roughly, this algorithm operates on an arbitrary scheme in the following way:

(a) Consider a synchronous scheme with no resets. We introduce first auxiliary variables and intermediate computations to make all terms simple. Next, we analyze the scheme by associating with each box in the scheme the following lists of bounded size:

(i) A list of the known truth values of tests.

(ii) Interrelations between variables. These will be equalities such as $y_1 = f(g(y_2,y_3),y_4)$. Note that we only have to retain such relations of depth not exceeding a bound dependent on k . Hence there are only finitely many such possible relations.

(iii) The difference in depth of each variable from the maximal depth. These will be numbers not exceeding k .

Obviously these lists can contain only a bounded amount of information, and correspondingly only finitely many different combinations of these lists exists. We use these lists as tags to the nodes in the scheme. We may now unwind the original scheme, tagging each box with the accumulated information. A node in the original scheme may correspond to several nodes in the new scheme, each tagged with its own annotation, providing full information about the interrelations between the variables. The resulting scheme will be simple, free, and synchronous.

(b) Next we consider a synchronous scheme with resets. Our task here is to eliminate the resets. Let us partition the scheme S into subschemes, $S_1,\ldots,S_n$, each of which contains a reset instruction as its first instruction, but no other instances of resets. There may be exits from any S_i to any S_j , which must go through the main entry of S_j .

We apply (a) above to each of the S_i 's. Then we form the cross product scheme $S' = S_1 \times S_2 \times \cdots \times S_n$ which jointly simulates $S_1,\ldots,S_n$. S' allocates disjoint sets of variables to each component S_i and performs in parallel the operations required by each of the components. A special tracing mechanism is provided for representing the situation that S_i exits to S_j . □

To decide the equivalence of two synchronous schemes, we first study the problem of inner equality. Since the result of the preceding algorithm tags each box with the interrelations between the variables, we only have to inspect the tag on all exit statements and check whether it contains (or implies) the statement $z_1 = z_2$. Now in order to test equivalence of S_1 and S_2 we have only to consider the inner equality problem for $S_1 \times S_2$.

Acknowledgements.

We are indebted to Richard Waldinger and Pierre Wolper for their detailed reading of this paper.

References.

[C] Chandra, A. K., "On the Properties and Applications of Program Schemas," Ph.D. Thesis, Stanford University, March 1973.

[CM] Chandra, A. K. and Z. Manna, "Program Schemas with Equality," Proceedings of the Fourth ACM Symposium on the Theory of Computing, Denver, Colorado, May 1972.

[G] Greibach, S. A., "Theory of Program Structures: Schemes, Semantics, Verification," Springer Verlag, lecture notes in Computer Science, 1975.

[I] Ianov, Y.I., "The Logical Schemes of Algorithms," Problems of Cybernetics, Vol. 1, Pergamon Press, New York (English translation), 1960.

[LPP] D. C. Luckham, D. M. R. Park and M. S. Paterson, "On Formalized Computer Programs," JCSS 4, 3 (June 1970), 220-249.

[M] Manna, Z., Mathematical Theory of Computation (Chapter 4), McGraw-Hill, New York, 1974.

[P] Paterson, M. S., "Equivalence Problems in a Model of Computation," Ph.D. Thesis, Cambridge, England, 1967.

[R] Rutledge, J. D., "On Ianov's Program Schemata," JACM 11, 1 (January 1964), 1-9.