Abstract State Machines and Pure Mathematics

Andreas Blass*

University of Michigan, USA
ablass@umich.edu

Abstract. We discuss connections, similarities, and differences between the concepts and issues arising in the study of abstract state machines and those arising in pure mathematics, particularly in set theory and logic. Among the topics from pure mathematics are the foundational role of set theory, permutation models of set theory without the axiom of choice, and interpretations (between theories or vocabularies) regarded as transformations acting on structures.

1 Introduction

The purpose of this paper is to describe some connections between the theory of abstract state machines (ASM's) and concepts from pure mathematics. These connections are of three general sorts.

First, there are direct uses of mathematical concepts in ASM's. A well known instance of this is the use of structures, in the sense of first-order logic, in the very definition of ASM's. A less well known instance is the use of interpretations, also in the sense of first-order logic, to describe transitions of ASM's as well as certain sorts of simulations.

Second, there are modifications of mathematical concepts, adapting them to the purposes of computation theory and ASM's. As an example of this, we discuss the ASM analog of the set-theoretic concept of permutation model.

Finally, there are analogies between ASM's and some aspects of pure mathematics. In this connection, we discuss the multifaceted philosophical issue of "universality": Do ASM's provide a universal framework for descriptions of algorithms in the same sense that set theory provides a universal framework for mathematical proofs? In this connection, we also discuss the value of explicit formalization and the role of definitions. We comment briefly on possible alternative "foundations" based on, for example, ordered lists or multisets instead of sets.

We also discuss the issue of "objects whose identity doesn't matter," which arises in connection with the `import` rules of ASM's and also in various contexts in pure mathematics.

* Preparation of this paper was partially supported by a grant from Microsoft Corporation. The opinions expressed here are, however, entirely my own.

Acknowledgements

I thank the organizers of the Monte Verità conference for inviting me to present the talk on which this paper is based. I also thank Yuri Gurevich for his suggestions for improving this paper and for many helpful and informative discussions about ASM's and other aspects of computer science.

2 Structures and Interpretations

From the very beginning, the concept of abstract state machines has been closely linked to pure mathematics. Indeed, one of the central ideas is to model states of computations as structures in the sense of mathematical logic. Another central idea is that an ASM program describes one step in a computation, not the entire iteration. Because of this, the programs themselves correspond to a familiar concept from first-order logic, that of interpretation.

Interpretations of one vocabulary or theory in another were introduced in [17] for the purpose of deducing undecidability of many theories from essential undecidability of one theory. See also [15, Section 4.7], whose formulation we follow here. In general, an interpretation of a vocabulary Υ in a theory T (with vocabulary Υ_T) consists of

- a unary predicate U in Υ_T,
- for each function symbol F of Υ a function symbol F_I of Υ_T, and
- for each predicate symbol P of Υ a predicate symbol P_I of Υ_T (equality does not count as a predicate symbol here but as a logical symbol),

such that it is provable in T that U is nonempty and closed under all the F_I's. Thus, given a model M for T, we obtain a structure for Υ by taking as its base set the extent in M of U and as its functions and predicates the (restrictions of the) interpretations in M of the corresponding symbols of Υ_T. The concept of interpretation is often extended by saying "interpretation in T" when one really means "interpretation in an extension by definitions of T." Then U, F_I, and P_I need not be primitive symbols of the vocabulary Υ_T but could be given by formulas in that vocabulary. (For historical accuracy, I should mention that the definition of "interpretation" in [17] already included extensions by definitions but did not include the unary predicate U; the latter was treated separately under the name of "relativization.")

Consider the special case of an interpretation where the theory T has no axioms (so the interpretation can be applied to arbitrary Υ_T-structures), where $\Upsilon = \Upsilon_T$, and where U is identically true (so applying the interpretation doesn't change the vocabulary or the base set of a structure). The transformations of Υ-structures obtainable by such transformations are just those obtainable by executing one step of an ASM with vocabulary Υ, provided the ASM doesn't create new elements and is non-distributed. (In the distributed case, the interpretations should be restricted so that the steps they describe can each be executed by a single agent.)

From this point of view, one could say that an ASM is "just a special sort of interpretation," but the word "just" here is unjust. Though formally the same, ASM's are intended to be used quite differently from interpretations: An ASM is to be applied iteratively (for a number of steps not usually known in advance). For interpretations in general, there is a notion of composition, but only interpretations from a vocabulary (or theory) to itself can be repeated arbitrarily often.

It may be noteworthy that, in the correspondence between ASM's and interpretations, the sequential ASM's correspond to quantifier-free interpretations. Thus, the same restriction is quite natural from both the computational and the logical points of view.

Can interpretations contribute anything to our understanding of ASM's? In a sense, they already did, at least if one considers interpretations in a generalized sense that has become common in model theory. There (following an idea that I believe was introduced by Shelah), one more or less automatically enlarges structures to include, in addition to their actual elements,

- tuples of elements and
- equivalence classes with respect to definable equivalence relations.

See for example the M^{eq} construction in [13, page 10]. Then interpretations can use these additional elements; thus for example the standard definition of the integers in terms of natural numbers (as equivalence classes of pairs) amounts to an interpretation. Some of the central ideas and results of [2] first appeared, in notes of Shelah, in the context of interpretations of just this sort. (They weren't explicitly called interpretations, but the influence of M^{eq} was visible.) Later, this material was clarified by expressing it in terms of ASM's. The distinctive set-theoretic context of [2] was motivated, at least in part, by the desire for a natural framework in which tuples and equivalence classes are easily handled.

Might interpretations contribute something more to ASM theory? What about interpretations between different vocabularies, which are therefore not iterable? Perhaps these will provide a useful point of view for such operations as setting up an initial state for an ASM. Often the input to a computation is only part of what should be present in the initial state of a computation. Adding the extra material to convert the input into the initial state is, in some cases, an interpretation in the generalized (M^{eq}) sense discussed above. In other cases, this preparation is more complicated, for example adding a whole set-theoretic superstructure as in [2]. It is not clear whether the notion of interpretation can be usefully stretched to cover such cases as well.

In addition to setting up the initial state for a computation, interpretations might be appropriate for extracting the "answer" from the final state of a computation. More generally, one can imagine many sorts of interfaces — not only input/output but more general interfaces between computations or between parts of a computation — as described by interpretations.

It also seems that interpretations may provide a good way to describe the connection between two ASM's that "compute the same thing in different ways." The idea here is that certain states, the "pose" states, in a run of the one machine

are obtainable from those of the other machine by a uniform interpretation. (Between successive pose states, each machine may go through states that have no strict counterpart in the other machine, states that depend on the details of how the machine does its work.) The interpretation tells how what one machine does is to be seen from the perspective of the other.

Finally, since interpretations are usually (in mathematical logic) regarded primarily as syntactic transformations rather than semantical ones, I should add a few words about the syntactic aspect. An interpretation of a vocabulary Υ in a theory T provides a translation of terms and formulas from Υ to the vocabulary of T. The translation consists of replacing each function symbol F or predicate symbol P in the given term or formula with F_I or P_I and restricting all quantifiers to U_I. Then the translation φ^I of a sentence φ is true in a model M of T if and only if the sentence φ itself is true in the Υ-structure M_I obtained from M via the interpretation.

Thinking of interpretations primarily on the semantic level, transforming models M of T into Υ-structures M_I, we can describe the syntactic transformation $\varphi \mapsto \varphi^I$ as producing weakest preconditions. That is, φ^I expresses, in a structure M, exactly what is needed in order that M_I satisfy φ. Here is a toy example to indicate what goes on in general.

Consider the update rule $c := d$. The corresponding interpretation has $P_I = P$ and $F_I = F$ for all predicate and function symbols other than c, and $c_I = d$ (and U_I is the identically true predicate). If M is a structure, then M_I for this interpretation is the same as M except that it gives c the value that d had in M, i.e., it is the sequel of M with respect to the rule $c := d$. Now consider a formula φ, the simplest relevant example being $P(c)$ for a unary predicate P. Its translation φ^I under our interpretation is obtained simply by replacing every occurrence of c by d, so in our simple example $P(c)^I$ is $P(d)$. And this φ^I is exactly what M must satisfy in order to guarantee that M_I satisfies φ.

3 Permutation Models

In this section, I'll describe a technique for describing and analyzing the idea of "not permitting arbitrary choices," both in set theory and in computation theory.

In 1922, Fraenkel [5] constructed models of set theory (with infinitely many urelements, also called atoms) in which the axiom of choice is false. The central idea is that all the atoms in a model of set theory "look alike"; more precisely, any permutation of the atoms induces an automorphism of the whole set-theoretic universe. Fraenkel formed the subuniverse of "sufficiently symmetric" sets, and showed that it satisfies all the usual axioms of set theory except the axiom of choice. ("Usual" is an anachronism here, since Fraenkel himself had just recently introduced the replacement axiom and given a precise formulation of Zermelo's separation axiom. But the axiom system, known as Zermelo-Fraenkel set theory, ZF, is certainly the usual one nowadays.) More precisely, Fraenkel's model consisted of those sets x which depend on only a finite subset F of the atoms,

in the sense that any permutation of the atoms fixing those in F will, when extended to an automorphism of the whole universe, also fix x. (Subsequently, other authors studied other notions of "sufficiently symmetric" in order to obtain independence results concerning weak forms of the axiom of choice, but these other notions will not be relevant here.)

A modification of Fraenkel's idea is at the heart of the main results of [2]. This paper introduced a rather liberal model of choiceless polynomial time computation. The model is an ASM whose initial state is obtained by building a universe of (hereditarily) finite sets over the input structure regarded as consisting of atoms; this makes many sorts of data structures available for the computation. In addition, parallelism is allowed both in `do-for-all` rules and in some set-formation constructions. It is shown, however, that such apparently simple things as the parity of the number of elements in an unstructured set cannot be computed in polynomial time, even in this liberal model, as long as arbitrary choices are prohibited. The proof of this theorem uses the hypothesis of choicelessness by deducing that, if the computation "uses" a set x, then it must also use all sets $\pi(x)$ obtainable from x by automorphisms of the set-theoretic structure, i.e., by permutations of the atoms of the input structure. The polynomial time hypothesis then ensures that there cannot be too many of these $\pi(x)$'s. The essential combinatorial step in the proof is a lemma allowing us to infer from "not too many $\pi(x)$'s" to x being fixed by all permutations that fix a certain number of atoms. In contrast to Fraenkel's situation, "a certain number" cannot be taken to be simply "finitely many," as there are only finitely many atoms altogether in the present situation. Instead, a quantitative estimate is needed, which depends on the ASM program under consideration. Thus, the "infinite vs. finite" dichotomy exploited by Fraenkel has been replaced by "large finite (growing as the input structure grows) vs. small finite (depending only on the ASM program).

A further refinement of Fraenkel's idea occurs in [14], where Shelah proves a zero-one law for properties of graphs computable in choiceless polynomial time. For the purposes of permutation arguments, there is a crucial difference between the unstructured (or the slightly more general "colored") sets considered in [2] and the graphs considered in [14]: Almost all finite graphs have no non-trivial automorphisms. So at first it would seem that symmetry arguments cannot be applied to computations taking random finite graphs as inputs. Shelah circumvented this problem by working not with automorphisms but with partial automorphisms of graphs. This required a rather delicate development, with careful attention to the sizes of the domains of these partial automorphisms, to ensure that the behavior of a choiceless polynomial time computation is invariant (in a suitable sense) under partial automorphisms of the input. For the (rather complicated) details, we refer the reader to [14] or to an easier to read (we hope) exposition that Yuri Gurevich and I are preparing.

4 ASM's and Set Theory

At a 1993 meeting in Dagstuhl, after hearing several talks about formulating various algorithms as ASM's (then still called "evolving algebras"), I made a comment along the following lines:

> It seems to me that expressing algorithms in the ASM formalism is rather like formalizing mathematical proofs in ZFC. At the beginning, one needs a number of examples of such formalizations, but after a while it becomes clear that any "reasonable" algorithm can be written as an ASM, just as any "reasonable" proof can be formalized in ZFC. And after a while, there is little point in actually carrying out the formalizations just in order to verify formalizability (in either situation) unless and until a genuinely problematic case arises.

In the case of sequential algorithms, it seems fair to say that this comment has been confirmed by subsequent experience. Better yet, it is confirmed by the main result of Gurevich [8], which asserts that any sequential algorithm can be expressed by an ASM provided it satisfies certain very natural postulates.

I'll comment later on the situation with non-sequential algorithms, but first let me point out some differences between the roles of ASM's and of ZFC even in the sequential case.

The most obvious difference is that most of us have never seen a non-trivial proof fully formalized in ZFC (nor would we want to see one), but we have seen non-trivial algorithms fully formalized as ASM's. ASM's are, by design, reasonably close to our mental images of algorithms; ZFC is similarly close to mental images of only small parts of mathematics (primarily set theory, and not even all of that). To write out the proof of, say, the commutative law for real addition in the primitive language of set theory would be a very time-consuming exercise for which I see no value at all.

In contrast, writing out explicit ASM programs for various algorithms is not only feasible but worthwhile. It has, for example, led to the detection of errors or ambiguities in specifications of programming languages and computer systems.

This distinction is reflected in a second difference between ASM and ZFC formalizations: the ubiquity of *definitions* in the development of mathematics on a set-theoretic basis. Such a development begins with axioms written in purely set-theoretic notation, but it soon introduces other concepts by definition, and it is not at all unusual for a concept to be many definitional layers removed from the primitive set-theoretic notions. Think, for example of the real numbers, defined as equivalence classes of Cauchy sequences of rational numbers. The definition of $\mathbb{R}$ sits at the top of a rather high tower (or pile) of other definitions.

Something roughly analogous happens in ASM's, in the idea of successive refinement, working from a high-level description of an algorithm down to a specific implementation. But the analogy is only a rough one for two reasons. First, the tower that connects low-level and high-level versions of an algorithm consists of ASM's at every level. The tower connecting ZFC to, say, the theory of partial differential equations, departs from ZFC formalization (in the strict

sense) as soon as one gets above the bottom level; the higher levels are formalized, if at all, in first-order theories obtained (formally) by adding definitions as new axioms. Second, the refinement process is a serious object of study in the ASM world, whereas in the set-theoretic world the process of definition is usually swept under the rug — so much so that many logicians would be taken aback by my comment a moment ago that differential equation theory is formalized not in ZFC but in an extension by definitions. They would probably say "what's the difference?" and accuse me of splitting hairs. The only authors I know of who have made a serious study of the role of definitions in set theory are Morse [12], who carefully analyzes the permissible syntax (far more general than what is usually considered), and Leśniewski [11] who actually uses definitions in a non-conservative way, as an elegant formulation of existence axioms.

This suggests a question for ASM theory: Can one push the refinement process as far into the background as set-theorists have pushed definitions (and if not then how far can one push it)? That is, can one arrange things so that all one has to write is a high-level specification and the sequence of "definitions" of its functions in terms of lower levels? The idea would be that the corresponding low-level ASM would either be produced automatically or would (as in the set-theoretic situation) be entirely irrelevant. And of course the high-level ASM and the definitions together should be significantly shorter than the low-level ASM they describe.

A third difference between ASM's and ZFC is the use of ASM's for detecting and correcting errors in intuitive specifications, programs, etc. As far as I know, formalization in ZFC has not played a role in detection and correction of errors in mathematical proofs. Errors certainly occur, and they are detected and corrected. (For particularly embarrassing examples, see the footnotes on page 118 of [9] and the introduction of [16]. I emphasize that the authors of [9] and [16] did not commit the errors but conveniently summarized them.) But the detection and correction seems to proceed quite independently of formalization. It is based on intuitive understanding and is therefore not very systematic. There are, of course, projects to formalize and systematically check, by computer, various parts of mathematics. But these seem to be of more interest, at the moment, to computer scientists than to typical mathematicians. The latter seem to be generally quite willing to rely on intuitive understanding rather than formal checking.

A fourth difference between the role of ASM's in formalizing algorithms and the role of ZFC in formalizing proofs is that in the latter context there is (to the best of my knowledge) no analog of [8]. That is, there is no general criterion guaranteeing that any proof, subject to some natural constraints, is formalizable in ZFC. Mathematicians other than set theorists generally take on faith that anything they would consider a correct proof can be formalized in ZFC. Set theorists are aware of possible difficulties (with universe-sized sets), but we are confident that we could recognize any such difficulty and determine whether a proposed argument really goes beyond ZFC (without actually writing out a formalization).

Having promised to comment on the situation for non-sequential algorithms, let me say that the situation there is less clear than in the sequential case. There is no analog of [8] (yet), and the world of parallel and distributed algorithms seems much more difficult to survey completely than the world of sequential algorithms. (Distinguishing between "parallel" and "distributed" in the sense of [7], it seems that an analog of [8] is within reach for parallel algorithms but not for distributed ones.)

A particular headache that has come up several times in my discussions with Yuri Gurevich is the question of cumulative updates. As a simple example, suppose we want to count the number of elements in some finite universe U (in an ASM). The natural way to do this is to have a counter, initialized to zero, which each of the relevant elements then increments by one. The problem is to do this in parallel, without arbitrarily fixing an order in which the various elements are to act. The rule

```
do for all v with U(v)
   c := c + 1
enddo
```

only increments the counter by one, no matter how many elements are in U. If we have an ordering of these elements, then we can let each one increment the counter in turn. But without an ordering, a genuinely parallel version of the algorithm seems to require something new.

The fundamental problem, though, isn't whether this or that situation calls for an extension of the framework but rather whether the framework can be completed at all. I am inclined to be optimistic: A few additions should cover all parallel algorithms. But I realize that my optimism may be the result of an overly naïve picture of the wild world of parallelism.

5 How Basic are Sets?

The title of this section is intended to refer to the computational (ASM) side of the picture, not the pure mathematical (ZFC) side. Of course, sets are basic in ZFC, but this is to some extent the result of an arbitrary choice. Indeed, Cantor, the father of set theory, seems to have regarded sets as intrinsically equipped with an ordering, so perhaps some sort of (generalized) lists would be more fundamental.

On the computational side, too, there is a tendency to view interdefinable concepts, like (finite) sets and lists, as having equal claim to being fundamental. But it is shown in [3] that this tendency is not always appropriate. When we do not allow arbitrary choices (or, equivalently, an ordering) but do allow parallelism, and when we impose polynomial bounds on the total computation time of all processors, then a set-theoretic environment as in [2] is strictly stronger than one using tuples or lists instead. (The polynomial time bound is essential here. Without it, parallelism could compensate for the absence of an ordering by exploring all orderings.)

This result suggests to me that a set-based environment is more appropriate for this sort of computation than the more familiar list-based environments. There is some tension here with the fact that lists can be more straightforwardly implemented than sets. But the sort of computation under consideration, with no ordering allowed, is already intrinsically removed from implementation, where an ordering is implicitly given by the machine representation of elements.

These considerations raise a further question concerning choiceless polynomial time computation: Might some other environment be even better than the set-based one? Perhaps the considerations at the end of the preceding section, about cumulative updates, suggest an answer, namely to use multisets and allow cumulative updating of multiplicities of membership in a multiset. In other words, allow updates of the form "throw element x into multiset y," with the convention that, if several parallel processes execute this update (with the same x and y), then the result is that the multiplicity of x's membership in y is increased by the number of these processes.

But can one do better yet? Might there even be entirely new data structures that are particularly useful in the choiceless context?

6 Unidentified Objects

In this final section, I'd like to comment on an issue that comes up in many contexts, throughout computer science and mathematics, but is almost always ignored because it's trivial. My main point is that, if it's so trivial, we should be able to give a clear explanation very close to our trivializing intuition.

The issue arises in the theory of ASM's in connection with the importing or creating of new elements. It doesn't matter what the new element is, as long as it's new. So we refuse to worry about the exact choice of the new element. (If two or more elements are to be imported simultaneously, then we make sure they're distinct, but beyond this avoidance of "clashes" we refuse to worry.) A mathematician's immediate reaction is that we work, not with a particular structure in which a particular element has been imported, but with an isomorphism class of structures, a different but isomorphic structure for each choice of imported element. This works, but it doesn't quite correspond to intuition; intuition is still dealing with one structure, not a whole class of them. In some sense, intuition deals with a "generic" member of the isomorphism class, but what exactly (mathematically) is that?

The same issue and the same mathematical solution occur in connection with the syntax of just about any formalized language. One uses bound variables, but one doesn't care what particular variable is used (as long as clashes are avoided). It has become customary to annihilate the issue by saying "we work modulo α-conversion," meaning that expressions differing only in the choice of bound variables are to be identified. This amounts to the "isomorphism class" viewpoint of the preceding paragraph. Another approach was taken by Bourbaki [4]. Their official syntax for logic and set theory (and thus for all of mathematics) has no bound variables. In their places are occurrences of the symbol $\square$. To

indicate which occurrences correspond to the same variable, these are joined to each other (and to the binding operator) by lines. This seems to me to be closer to intuition, but rather unreadable. And it applies only to syntactic situations; I don't see how to adapt it to a semantic situation like ASM's. (I find it somewhat reassuring that the highly respected mathematicians of Bourbaki — or at least some of them — found this issue to be worth thinking about to the extent of producing a rather unorthodox solution.)

Another approach to this issue, in the syntactic context of bound variables, is the use of de Bruijn indices. These indices link a variable-binder, such as a quantifier, with the occurrences of the variables it binds, by specifying the difference between them in quantifier depth. That is, in place of a Bourbaki box with a line joining it to a quantifier, one would have a number indicating that that the relevant quantifier is to be found so and so many levels less deep in the parse tree. This notation strikes me as farther from intuition than Bourbaki's boxes but closer than isomorphism classes. In terms of human readability, it seems no better than the boxes, but I understand computers read it quite well. (Perhaps we humans just need to be wired differently.) I see no way to adapt this approach to non-syntactic situations, like the choice of new elements created in an ASM.

A variant of the isomorphism class approach is suggested by the topos-theoretic view of generic or variable objects [10]. In the present context, this amounts to regarding the isomorphism class not simply as a class but as an indexed or parametrized family, the parameters being the individual choices. Thus, for instance, an ASM that imports two elements would be viewed as a family of ASM's indexed by ordered pairs of distinct elements. The ASM indexed by the pair (x, y) is the one in which first x and then y were imported. An elegant framework and notation for this approach (applied to bound variables) is given in [6].

The issue of unidentified objects arises in many other contexts. Indeed, one can claim that the exact identity of mathematical objects never matters; all one cares about is the structure relating them. For example, it never matters whether real numbers are Dedekind cuts or equivalence classes of Cauchy sequences, as long as they form a complete ordered field. The following quotation from [10, page 119], though intended to emphasize the topos theorist's view of sets in contrast to the set theorist's cumulative hierarchy, seems to accurately describe how most mathematicians view sets.

> An abstract set X has elements each of which has no internal structure whatsoever; X has no internal structure except for equality and inequality of pairs of elements, and has no external properties save its cardinality; still an abstract set is more refined (less abstract) than a cardinal number in that it does have elements while a cardinal number does not.

This comes very close to saying that the actual elements don't matter as long as there are some (and equality is determined). But I claim that we have no

entirely satisfactory semantics for dealing with the concept of an arbitrary object whose actual identity doesn't matter (but whose distinctness from other arbitrary objects may be important).

Notice that the problem I am raising is a semantical one. Syntactically, we can handle objects whose identity doesn't matter: just add constant symbols to the language to serve as names for the desired objects. This process is well understood in the context of first-order logic (see for example [15, Chapter 4]; it plays a crucial role in the proof of Gödel's completeness theorem). But passing from syntax to semantics requires choosing elements to serve as the denotations of the new constant symbols, and that brings us right back to the original problem: We need to choose an element but without caring which element it is.

Finally, let me mention that this problem threatens to become more acute if one considers quantum computation or indeed any quantum phenomena. In the quantum world, actual physical objects, like elementary particles, have identities in only a very limited sense. It makes sense to talk about an electron and another electron, but not to talk about this electron and that one — interchanging the two electrons in some physical process yields not another process but another channel for the same process, and two channels may interfere (constructively or destructively).

A Appendix on Defaults and Environment

In this appendix, we collect some observations and questions about ASM's whose connection to pure mathematics is even more tenuous than in the last section. Indeed, most of them are based on a contrast rather than a connection between the computational and mathematical worlds.

A major difference between the viewpoints of ASM's (together with most of computer science) and pure mathematics is that in the former a dynamic aspect is always present. An ASM doesn't just sit there, it undergoes transitions — as the old name "evolving algebra" emphasizes. Of course, pure mathematics is also capable of dealing with dynamic situations, but this is always explicitly emphasized, not part of a universal background as in ASM's.

There is more to the dynamic aspect of ASM's than just that their dynamic functions can change their values. It is also important that dynamic functions retain their previous values unless explicitly changed. Ironically, this apparently simple default assumption, favoring persistence of values, can increase the logical complexity of what an ASM does. For example, the action of a parallel rule

```
do for all v
   R(v)
enddo
```

could be described in existential logic (or in the existential fixed point logic advocated in [1]) provided its body $R(v)$ could be so described. The parallel rule executes an update if and only if *there exists* a value of v for which $R(v)$ executes that update. But the next state cannot be described existentially, because the

inaction of the rule, the persistence of dynamic functions when not updated, requires a universal quantifier for its description.

The fact that parallel ASM's can be limited to quantifier-free guards, using `do for all` to simulate quantifiers, is ultimately due to the presence of an implicit universal quantifier in the default requirement that values persist unless explicitly changed. (In the last two paragraphs, I've pretended for simplicity that the ASM's under consideration never attempt conflicting updates. The possibility of clashes would introduce additional universal quantifiers, because an update is executed if some $R(v)$ would execute it (ignoring clashes) and *there are no* w_1 and w_2 for which $R(w_1)$ and $R(w_2)$ would execute conflicting updates.)

Another sort of default adds complexity not in the logic but in the computational aspects of ASM's. This default is the assumption that, when new elements are created (or imported from the reserve), they arrive with no structure: All Boolean functions, except equality, produce `false` when one of their arguments is a freshly created element; all non-Boolean functions produce `undef` under these circumstances. Thus, for example, if P is a binary Boolean function and x is a newly created element, then $P(x, y)$ miraculously has the value `false` for all y. If we ask how it got that value, there are several possible viewpoints. One is that creating an element really means importing it from the reserve, and that the appropriate default values were already there while x was in the reserve — so the default value of $P(x, y)$ for newly imported x is just a matter of the persistence default discussed above. But of course this viewpoint requires that the initial state of a computation include the appropriate default values for reserve elements, an assumption that is appropriate only at a sufficiently high level of abstraction. At a lower level, one would have to ask how this initialization is to be performed. Another viewpoint is that the defaults are (at least implicitly) set by the ASM at the time the new element is created. This amounts to a fairly large scale parallel operation, not available in the sequential situation; it may be the approach closest to what happens in actual computers when new memory locations are allocated to a computation. A third viewpoint is that the setting of the defaults is, like the choice of which element to import, the responsibility of the environment. This seems to be the simplest approach, but it strikes me as a bit unfair to the environment. Making the environment choose the new element is reasonable, because this cannot be accomplished algorithmically (unless an ordering or some similar structure is available); but setting the defaults could be done algorithmically (in the case of parallel ASM's) so the justification for turning the job over to the environment seems to be only that it's more work than the ASM wants to do.

Let me close with a brief comment, related to the preceding only because it's about the environment. The environment of an ASM is used to model a variety of things: the choice of elements to import (or create), the arbitrary choices involved in non-determinism, input-output operations, and, in distributed computing, all the agents other than the one under consideration. How similar are these, really? There is of course one similarity, which caused them to all be called "environment" in the first place: They are not part of the algorithm (ASM)

under consideration, but they interact with it. Is there further similarity among some (or all) of these aspects of the environment? Are there *useful* distinctions to be made? ("Useful" means, at a minimum, more useful than just listing the various items as I did above.)

One rather imprecise but perhaps useful distinction is obtained by singling out those aspects of the environment's activity that one would expect to be included in a system for executing ASM programs. Such a system should not be expected to provide input or to perform the actions of distributed agents other than the one being simulated. But it could reasonably be expected to execute `import` rules on its own, or at most with the help of the operating system under which it runs.

References

1. Andreas Blass and Yuri Gurevich, "Existential fixed-point logic," in *Computation Theory and Logic*, ed. by E. Börger, Lecture Notes in Computer Science 270, Springer-Verlag (1987) 20–36.
2. Andreas Blass, Yuri Gurevich, and Saharon Shelah, "Choiceless polynomial time," *Ann. Pure Appl. Logic*, 100 (1999) 141–187.
3. Andreas Blass, Yuri Gurevich, and Jan Van den Bussche, "Abstract state machines and computationally complete query languages," this volume.
4. Nicolas Bourbaki, *Elements of Mathematics. Theory of Sets*, Hermann (1968).
5. Abraham Fraenkel, *Der Begriff "definit" und die Unabhängigkeit des Auswahlaxioms*, Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-Mathematische Klasse (1922) 253–257.
6. Murdoch J. Gabbay and Andrew M. Pitts, "A New Approach to Abstract Syntax Involving Binders," in *Proceedings 14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 1999*, IEEE Computer Society Press (1999) 214-224.
7. Yuri Gurevich, "Evolving Algebra 1993: Lipari Guide", in *Specification and Validation Methods*, ed. by E. Boerger, Oxford University Press, 1995, 9–36.
8. Yuri Gurevich, "Sequential abstract state machines capture sequential algorithms," *ACM Transactions on Computational Logic*, to appear.
9. Thomas Jech, *The Axiom of Choice*, North-Holland (1973).
10. F. William Lawvere, "Variable quantities and variable structures in topoi," in *Algebra, Topology, and Category Theory (A Collection of Papers in Honor of Samuel Eilenberg)*, ed. by A. Heller and M. Tierney, Academic Press (1976) 101–131.
11. Eugene C. Luschei, *The Logical Systems of Leśniewski*, North-Holland (1962).
12. Anthony P. Morse, *A Theory of Sets*, Academic Press (1965).
13. Anand Pillay, *Geometric Stability Theory*, Oxford University Press (1996).
14. Saharon Shelah, *Choiceless polynomial time logic: Inability to express*, paper number 634, to appear.
15. Joseph Shoenfield, *Mathematical Logic*, Addison-Wesley (1967).
16. Edward E. Slaminka, "A Brouwer translation theorem for free homeomorphisms," *Trans. Amer. Math. Soc.*, 306 (1988) 277–291.
17. Alfred Tarski, Andrzej Mostowski, and Abraham Robinson, *Undecidable Theories*, North-Holland (1953).