Rich Sequential-Time ASMs

Yuri Gurevich, Wolfram Schulte, and Margus Veanes

Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
{gurevich, schulte, margus}@microsoft.com

1 Introduction

Software industry can use a good specification language. This specification language should be executable, and so ASMs become relevant. The language should allow one to integrate specifications with existing technologies. To meet this need, our group in Microsoft Research builds a powerful extension of the original ASMs. We call it ASML (for "ASM Language"). ASML has a rich type system, is object oriented, and is being integrated with Microsoft programming environment. Here we are not concerned with the precise syntax of ASML. We view the original ASMs as mathematical objects and we extend the theory of original ASMs to provide a solid semantic foundation for ASML. The explicitly distributed version of ASML is yet to be implemented; accordingly we deal only with sequential-time computing in this paper.

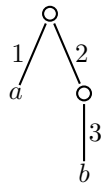
2 Extension Layers

Our starting point is the ASM model of the Michigan guide [2]. We extend that model with rich background structure (of the kind introduced in [1]) which includes (finite) maps. Then we introduce creation of new elements *together* with initialization. This allows us to introduce classes and objects in a natural and easy way. After that we merge expressions and rules into one syntactic category: *rule expressions*, or *rexes*. A rex may return a value and may produce updates. Finally we introduce exception handling and subASMs. We do not introduce typing; it is not needed for semantical purposes.

3 Maps

Maps are convenient. They allow us to view dynamic functions as elements (namely maps) and therefore treat finite dynamic functions of positive arity as nullary. This convenience comes at a price. Consider a map μ whose range contains a map $\mu(a)$ whose range contains a map $\mu(a)(b)$, and so on. This cannot go forever because μ is finite, but it can go for a while. Now imagine that you want to update μ at a, and update $\mu(a)$ at b, and so on, all in the same time. This poses a challenging update-consistency problem which is solved in the paper.

We illustrate the difficulties with an example. It is useful to depict a map μ as a (finite) tree with an immediate subtree that is a depiction of $\mu(x)$ for each element x in the domain of μ. Let f be a dynamic function whose value is the map $\{1 \mapsto a,\ 2 \mapsto \{3 \mapsto b\}\}$, f can be depicted as



Suppose that we wish to update $f(2)$ to the map $\{5 \mapsto c\}$, by firing the rule

$$\mathtt{f(2)} := \{\mathtt{5} \mapsto \mathtt{c}\}. \tag{1}$$

Moreover, suppose that we wish to update $f(2)(3)$ to d. To this end we may use the rule

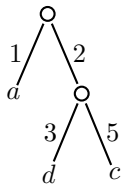
$$\mathtt{f(2)(3)} := \mathtt{d}. \tag{2}$$

If both rules are fired in parallel then the resulting update is clearly contradictory, because, according to the first rule the value of $f(2)$ in the sequel of the given state is the map $\{5 \mapsto c\}$ and thus the value of $f(2)(3)$ undefined, and according to the second rule, the value of $f(2)(3)$ in the same state is d.

If we instead of firing the rule (1) fire the rule

$$\mathtt{f(2)(5)} := \mathtt{c} \tag{3}$$

in parallel with the rule (2) then the resulting update set is consistent and the value of f in the sequel of the given state is, as expected, the map



3.1 Update Inconsistency as an Exception

The rich world of maps allows us to represent updates as values for expressions involving appropriate map operations. Those operations check that their arguments are compatible and if not yield an exception indicating inconsistency. As a result, the exception handling mechanism can treat update inconsistency as just another exception.

3.2 The Initialization Problem

You want to import a new element from the reserve and at the same time initialize certain dynamic functions at the new element. But you don't want to have a special initialization step, and of course you don't want to change the state in the middle of a step. What can you do? The solution is facilitated by the background structure. We explain the solution on a simple example. Suppose that you want to import a new element x and set $\texttt{color}(x)$ to `white`. Let p be the ordered pair of elements (denoted by) `color` and `white`. The pair $\langle x, p\rangle$ exists in the background structure. Put it into a special depository. Consult the depository whenever you need the attribute `color` of x.

4 A Pleasant Surprise

It turns out that ASMs, enriched with background and initialization depository, fit nicely to deal with classes and objects; compare this with [3]. Indeed what is a class C? First of all, the name C is a nullary dynamic function. In a legal state, the interpretation of C is a set of atoms (in the sense of the underlying background structure) which is typically empty in the beginning of the computation. The objects (or instances) of C are elements of that set. An instance field F of a class C has an implicit argument in addition to the explicit parameters that it can have; this implicit argument is an instance of C. A method M of C is a named rule with a similar implicit argument.

References

1. Andreas Blass and Yuri Gurevich. Background, reserve, and Gandy machines. In Peter Clote and Helmut Schwichtenberg, editors, *Computer Science Logic: 14th International Workshop*, volume 1862 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2000.
2. Y. Gurevich. May 1997 draft of the ASM guide. Technical Report CSE-TR-336-97, EECS Department, University of Michigan, 1997.
3. A. V. Zamulin. Generic facilities in object-oriented ASMs. In Y. Gurevich, P. W. Kutter, M. Odersky, and L. Thiele, editors, *Abstract State Machines: Theory and Applications (ASM'2000)*, volume 1912 of *Lecture Notes in Computer Science*, pages 91–111. Springer, 2000.