ASMs in the Classroom: Personal Experience*

Yuri Gurevich
Microsoft Research
One Microsoft Way, Redmond, WA 98052

Abstract

We share our experience of using abstract state machines for teaching computation theory at the University of Michigan.

1 Introduction

Dines Bjørner asked me to write a short non-technical essay "taking its departure" in the chapter Abstract State Machines for the Classroom by Wolfgang Reisig [6]. Well, I like Wolfgang's chapter very much. Let me use this opportunity to share some of my experience of teaching with ASMs at the University of Michigan. I was at Michigan from 1982 till 1998, most of the time (from 1984 on) with the Department of Electrical Engineering and Computer Science (EECS). The last few of those Michigan years I used ASMs in my teaching. To keep this essay short, let me restrict attention to the course on computation theory.

I taught the course often. At the Mathematics Department of Israel's Ben Gurion University, where I taught before coming to Michigan, undergraduate courses were up for grabs, and I enjoyed teaching and learning new courses. In EECS, the undergraduate curriculum was partitioned into feudal domains, and the small computer theory group owned few courses. Kevin Compton, my fellow theorist in EECS, said once: "I've taught that course so many times that I could do it in my sleep . . . and often have." In this connection, I tried each time a new angle in teaching the course, which partially explains why my frequent teaching of the course did not result in a book. Since 1998, I am with Microsoft Research. The engineering culture of Microsoft has rubbed off on me, and today my teaching would be different. But I would continue to use ASMs in my teaching; my confidence in ASMs has only grown.

The computation theory course was a part of the official curriculum of the Association for Computing Machinery (ACM), and it served as a prerequisite for some other courses. It was supported by the venerable 1979 "Introduction to Automata Theory, Languages, and Computation" by John Hopcroft and Jeffrey Ullman [4], and new excellent textbooks kept appearing. Nevertheless in

*To appear in Springer's Logic of Specification Languages [2]

the 1990s the course on finite state machines, pushdown automata and Turing machines seemed antiquated. Computing became so much broader: graphical user interfaces, parallel and distributed computing, networks, Web based computing and searching, communication and security protocols, and other forms of computing that didn't exist or weren't yet important in 1979. And computer science attracted more students than ever before, many of them mathematically challenged. Students (and their parents!) complained that they don't see much use for Turing machine programming. ASMs could describe arbitrary computations naturally and *without lowering the abstraction level*. That gave me an idea of using ASMs for teaching. But I had to be careful in fiddling with the computation theory course; it was a traditional undergraduate course, a part of the ACM curriculum and a prerequisite for other courses.

Acknowledgments

Many thanks to Andreas Blass and Kevin Compton for their comments on a draft of this essay.

2 Finite state machines and context-free languages

The computation theory course consisted of three parts: (i) finite automata and regular languages, (ii) context-free languages and pushdown automata, and (iii) Turing machines, undecidability and complexity. The first two parts were similar from the point of view of the use of ASMs. They included numerous algorithmic constructions, for example the subset construction that turns a nondeterministic finite automaton into an equivalent deterministic one. The constructions would typically involve parallelism. You can write prose describing such a construction or give pseudocode for the construction. I programmed the constructions as ASMs and required the students to do the same. The ASM programs looked like pseudocode of a particular style to the students. But we were using of course a precise ASM computation model which allowed us to use strings as well as finite sets and sequences of arbitrary entities. A couple of ASM programs from that computation theory course appear, slightly modified, in [3, Section 3].

Typically the students would adapt to the "pseudocode" style rather quickly. One difficulty was related to the default parallelism of ASMs. In conventional programming languages, like C, commands listed in some order are executed in that order. In the ASM world, the default is parallel execution. If you want that commands be executed sequentially, you have to say so explicitly [3]. The students had programming experience in conventional languages and found default parallelism strange. I could circumvent the difficulty by making parallelism explicit and writing "do in parallel" in the appropriate places but I didn't want to clog the ASM code. More importantly, I wanted to break the sequential-by-default thinking and thus elevate the abstraction level of programming.

The primary beneficiary of the ASM use in parts one and two of the computation theory course were mathematically challenged students scared of proofs. Programming was a different story. It was often their strong suit. ASMs allowed me to present proof assignments as largely programming assignments without lowering the abstraction level and with no unnecessary details.

Today I would use AsmL, the high level specification/programming language developed in Microsoft Research [1], so that the students could execute their programs. (I would also show the students finite and pushdown automata that do something useful, e.g. the scanner of a lexical analyzer and a mini parser respectively.)

3 Universal computation models

It is in the third part of the computation theory course that we really took advantage of ASMs. I would tell the students that they are already familiar with a universal computation model. The dramatic effect was lost on some as they had read about Turing machines or realized where I was going. But there were always some students who looked perplexed and were about to protest. At this point, I would explain that ASMs constitute a universal model, and we had been using the ASM computation model all along. I never dwelt on the greater universality of ASMs. That aspect of ASMs was beyond the scope of the course.

Of course Turing machines (TMs) were introduced as well. They are indispensable for undecidability proofs. So both ASMs and Turing machines were used. And there was a price to pay: to show that TMs can simulate ASMs. (The other way round is obvious). This was done via random access machines (RAMs). First show that TMs can simulate RAMs, and then show that RAMs can simulate ASMs. There are elegant forms of the first simulation in the literature, see [5] for example. The second simulation was recently simplified and made elegant in [7].

The price was worth paying. ASMs allowed us to avoid Turing programming. Consider for example the theorem that, for every nondeterministic TM that computes a function, there is a deterministic TM that computes the same function. The idea is simple — interleave all possible computations of the given nondeterministic TM, but the TM programming of the interleaving is a tedious task on a low abstraction level. Instead program the interleaving in the ASM language and then refer to the fact that TMs can simulate ASMs; the ASM programming of the interleaving is a programming exercise that can be assigned as a part of home work. Even proving an initial undecidability result is made easier by the use of ASMs.

Another advantage of using ASMs was that, contrary to Turing machines, ASMs could be actually used for various purposes. In fact, throughout the course, we used ASMs to program algorithms, many of them highly parallel. The reader may ask why not use a conventional programming language instead of ASMs for programming algorithms. The reason is that ASMs allow you to program algorithms without lowering the abstraction level.

References

[1] Abstract State Machine Language (AsmL), `http://research.microsoft.com/fse/asml/`

[2] *Logic of Specification Languages*, Editors Dines Bjørner and Martin C. Henson, Springer Texts in Theoretical Computer Science (An EATCS Series), to appear.

[3] Yuri Gurevich, Margus Veanes and Charles Wallace, "Can Abstract State Machines be Useful in Language Theory," Technical report MSR-TR-2006-159, Microsoft Research, 2006.

[4] John E. Hopcroft and Jeffrey D. Ullman, "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, 1979.

[5] Christos H. Papadimitriou, "Computational Complexity", Addison-Wesley, 1994.

[6] Wolfgang Reisig, "Abstract State Machines for the Classroom", a chapter in [2].

[7] Comandur Seshadhri, Anil Seth and Somenath Biswas, "RAM Simulation of BGS model of Abstract State Machines?" *Fundamenta Informaticae*, Selected papers from ASM 2005, to appear. (Full Proc. ASM 2005 are found online.)