# Teaching ASMs, Teaching with ASMs: Opportunities in Undergraduate Education

James Huggins[1], Jean Mayo[2], and Charles Wallace[2]

[1] Computer Science Program, Kettering University
Flint, Michigan, USA
[2] Computer Science Department, Michigan Technological University
Houghton, Michigan, USA

Proponents of Abstract State Machines (ASMs) often cite their "easy learning" and low degree of "formal overhead" as points in favor of their use by systems developers. Recent achievements in automated tool support lend further credibility to ASMs as an industrial-strength methodology. Yet there has been relatively little written on the empirical arguments in favor of ASMs. *Do* people truly find ASMs easy to understand and use? As educators, we are particularly interested in how readily this new technology transfers to the classroom. Furthermore, we would like to take it beyond courses like software engineering, where formal methods are commonly encountered, and extend it to other areas. As many experts have pointed out, the only way in which advanced software engineering tools and methods can make headway into popular software design is if they are woven into the fabric of traditional computer science courses. We report on our current work in bringing ASMs to the classroom.

**Instructional materials for ASMs.** Over the years, various introductions to ASMs have been written. In addition, many ASM papers include brief overviews for the casual reader. While many of these are well written and illuminating, we find they are not an ideal match for our audience. Our students are talented and experienced in finding solutions, but they often find it difficult to reflect on their "internalized design process" and document the design decisions they have made. They need reassurance that it is legitimate, and evidence that it is beneficial, to draw up designs that leave some details unspecified.

In the tradition of Gurevich's original ASM tutorial and subsequent works, our ASM primer is a dialogue between the authors and an inquisitive undergraduate student, Questor. This style adds a certain degree of levity to what can be a rather dry exposition. It also allows us to empathize with the student readers, addressing issues they typically find difficult as they arise. From experience, we know where the obstacles to understanding lie: the notion of functions that change meaning, the parallelism of block rules, the representation of complex computations simply as static functions. Our current primer covers only sequential examples: greatest common divisor, string matching, and minimum spanning tree. We plan a series of such primers. The next will delve into a "real-world" example: the database recovery problem addressed in an earlier ASM paper.

**ASMs in a software engineering curriculum.** As part of the new Software Engineering degree option, a new course has been added, focussing on aspects of the software process concerned with quality assurance: requirements, specification, formal description and modeling, and verification (mainly black-box testing). The course starts with an investigation of "problem structuring and analysis" through *problem frames*, which provides a good foundation for the subsequent coverage of ASMs.

**ASMs in a concurrent computing curriculum.** We intend to promote the use of abstraction, information hiding, and refinement in the design of concurrent software by incorporating ASMs into the teaching of concurrent computing. Our plans include:

- Providing *automated tool support* by incorporating the *XASM* tool into existing *ConcurrentMentor (CM)* visualization software. Students will be able to write software partly in C++, using the class libraries of CM, and partly in ASM. The ASM portions will be compiled to C++, using a modified version of XASM's ASM-to-C compiler. CM provides a C++ library of communication primitives and visualization capabilities. Students will benefit from the expressivity of ASM and the visualization and analysis features of CM. The movement from abstract design to concrete implementation will be achieved by gradually replacing ASM code with equivalent C++ code.
- Creating a *corpus of documented examples* for using ASM in the design of concurrent programs. This will consist of an online library of collections of ASM and C++ code.