8 Tool Support for ASMs

In this chapter we discuss the various forms of tool support for the analysis of ASMs, namely by mechanical verification systems and by environments to refine ASMs into executable programs one can use for validation purposes.

8.1 Verification of ASMs

In the preceding chapters numerous proofs have been given to illustrate how one can verify dynamic properties for given ASMs by exploiting the fact that ASMs are not logical formulae, but machines coming with a notion of run which lends itself to traditional inductive arguments. The proofs range from simple to more challenging ones and are of a mathematical nature, providing for human experts, of the considered application domain or of software design, arguments which are deemed to be rigorous and complete enough to represent a verification of the claims of interest. Certainly such proofs provide no absolute guarantee, they may be incomplete or contain flaws. If the complexity is not prohibitive, the proofs may be detailed further to become still more trustworthy, e.g. by formalizations in appropriate logical calculi or for mechanical machine-supported verification. This will entail a considerably higher effort and cost, which the development manager should be aware of; see the evaluation in [65] and the positive experience with the industrial use of the B method [37].

There is no obstacle of principle to applying mechanical theorem proving and model checking systems to verify the properties of ASMs. In fact KIV, Isabelle, PVS and model checkers have been successfully used in this way (for verifying correctness properties for compilers, architectures, protocols, control programs etc., as has been pointed out in the preceding chapters) and numerous logics have been developed to deal with specific features of ASM verification (see Sect. 9.4.3 for detailed references). In Sect. 8.1.1 we present the unifying logic developed in [405], which is tailored for ASMs in terms of an atomic predicate for function updates (together with a definedness predicate for the termination of the evaluation of turbo ASMs). This chapter can be read independently of the others except for the definition of ASMs in Sect. 2.4, but the reader is supposed to have a basic knowledge of first-order
logic. In Sect. 8.2 we briefly explain the basic transformation of ASMs to FSMs which underlies the applications of model checking to ASMs.

8.1.1 Logic for ASMs

The logic for ASMs that we describe in this section differs from other logics that have been proposed for ASMs in two points: (i) it is complete for the class of ASMs that do not contain cycles in the dependency graph of rule declarations (so-called hierarchical ASMs); (ii) it can be applied also to turbo ASMs where the evaluation of transition rules might not terminate.

The reader may wonder why a logic for a computationally universal mechanism like hierarchical ASMs can be complete at all? The answer is that the logic is not able to talk about full ASM runs. Instead, the logic talks about single steps of an ASM and therefore the logic is complete for statements about the single steps of an ASM like the invariants of rules, the consistency conditions for rules, or the step-for-step equivalence of rules. Moreover, the completeness theorem holds for the uninterpreted logic, where the static functions do not have a fixed standard interpretation. Hence, the logic is complete in the same sense as first-order logic (FOL) is complete. In fact, it turns out that the logic for ASMs is a definitional extension of FOL. This means that the formulas of the rich language of the logic for ASMs (including the modal operators and a special predicate for function updates) can be translated into pure first-order formulas.

If we allow cycles in the dependency graph of rule declarations (as it is allowed for turbo ASMs), then the logic cannot be complete, since iteration and while-loops can be recursively defined and therefore it is possible to talk about the outcome of full ASMs runs. The same arguments, that are used to show that the dynamic logic is not complete, can be applied in this case.

Since ASMs are special instances of transition systems, the logic contains modal operators. We do not, however, stick to modal logic, since in some situations it can be more convenient and even more economical to use functions with an additional argument that specifies the $n$th state of the run of an ASM (explicit time versus implicit time). This has been shown to be useful in correctness proofs of ASM refinements (see Sect. 3.2.2).

What comes closest to the logic is known as dynamic logic with array assignments [268, 269]. In the dynamic logic with array assignments, the programs are sequential while-programs that manipulate arrays. There is no notion of parallel execution. Hence, the dynamic logic with array assignments is not concerned with the parallel execution of assignments and therefore does not need a notion of consistency for programs. The substitution principle which is used in its axiomatization is derivable (see Lemma 8.1.11).

For reasons that we explain below in Sect. 8.1.4 we exclude the choose construct from the logic. This means that the transition rules that are allowed in ASMs of the logic are deterministic. It does not mean that everything has to be deterministic. The logic can still be used to prove the properties of