ACM Computing Surveys 28(4es), December 1996.

Logics for Concurrency: Structure Versus Automata

Faron Moller

Abstract

We discuss a dichotomy within the study of temporal logics for reasoning about concurrent systems. This dichotomy is generally recognized as one between *linear-time* and *branching-time* logics. We propose that the division can also be drawn in terms of *global* versus *local* verification techniques, which roughly corresponds to a division between *automata-theoretic* and *algebraic-structural* techniques. We also suggest that the division can be defined – to an extent – geographically, and that these divisions roughly coincide.

There has been a great deal of effort spent on developing methodologies for specifying and reasoning about the logical properties of systems, be they hardware or software. Of particular concern are issues involving the verification of safety and liveness conditions, which may be of mammoth importance to the safe functioning of a system. While there is a large body of techniques in existence for sequential program verification, current technologies and programming languages permit ever greater degrees of concurrent computation, which make for a substantial increase in both the conceptual complexity of systems and the complexity of the techniques for their analysis.

One fruitful line of research has involved the development of process logics. Although Hoare-style proof systems have been successful for reasoning about sequential systems, they have had only minor impact when primitives for expressing concurrent computation are introduced. Of greater importance in this instance are, for example, dynamic, modal, and temporal logics. Such logics express capabilities available to a computation at various points of time during its execution. Two major varieties of such logics exist: linear-time logics express properties of complete runs, whilst branching-time logics permit a finer consideration of when choices within computation paths are made. The study of such logics is now quite broad and detailed.

However, a perceived dichotomy exists in temporal logic research that is defined largely by these two varieties of logics. In the *Strategic Directions in Computing Research Concurrency Working Group Report* [1], this dichotomy is defined directly in terms of the linear-time/branching-time divide. This divide, however, also appears to be defined to a large extent geographically (though, as with all things, this division is far from definitive).

On the one hand, the North American approach has exploited automata-theoretic techniques that have been studied for decades. The benefit of this is the ability to adopt theoretical techniques from traditional automata theory. Hence, for example, to verify that a particular system satisfies a particular property, one would construct an automaton which represents in essence the cartesian product of the system in question with the negation of the property in question, and thus reduce the problem to one of checking emptiness of the language generated by an automaton. This approach is particularly applicable to the verification of linear-time properties.

One obvious drawback to this "global" approach is the overhead incurred in constructing such an automaton: even if the system being analysed can be represented by a finite-state automaton, this automaton will typically be of a size exponential in its syntactic description (due to the so-called state-space explosion problem), and hence this approach to the verification problem is (at least) exponential in the worst case. Of course, this approach is inapplicable to infinite-state systems unless you first introduce symbolic techniques for encoding infinite sets of states effectively.

On the other hand, the "Eurotheory" approach to the verification problem is based more on exploiting structural properties. The essence of this approach is to work as far as possible with the syntactic system and property descriptions rather than their semantic interpretations as automata. Typically this involves the development of tableau-based proof systems that exploit congruence and decomposition properties. This allows a more "local" model-checking technique that could feasibly handle even infinite-state systems, as the overhead of constructing the relevant semantic automata is avoided.

Whichever approach is considered, there are common goals that are stressed. First and of utmost importance, the methodology must be faithful to the system descriptions being modeled as well as to the intended properties being defined. The methodology must be sound with respect to its intended purpose. Of great importance as well is the ability to automate the techniques; as economically- and safety-critical systems being developed become more and more complex, the need for automated support for their verification becomes an imperative of ever greater importance. Mechanical procedures for carrying out the verification of such properties are required due to the sheer magnitude of the systems to be verified. Beyond this, such algorithms must be tractable. There will clearly be trade-offs between the expressivity of a formalism employed to define system properties and the complexity of the verification of these properties. The verification procedures, beyond being sound, must be effective as well as of acceptable time and space complexity.

We thus have the following important themes appearing in the ongoing research into logics for concurrency:

- What are suitable formalisms for expressing system properties ?
- How expressive are these formalisms ?
- How difficult are they to employ, both conceptually and computationally ?
- What are the advantages and drawbacks when comparing formalisms ?

These themes are explored in detail in a collection of tutorials published in [2].

References

[1] R. Cleaveland and S.A. Smolka (editors), Strategic Directions in Concurrency Research. ACM Computing Surveys **28**(4), December 1996.

[2] F. Moller and G. Birtwistle (editors), **Logics for Concurrency: Structure versus Automata**. Springer Lecture Notes in Computer Science **1043**, 1996.