12

MODAL MU-CALCULI

Julian Bradfield and Colin Stirling

1 INTRODUCTION

Modal mu-calculus is a logic used extensively in certain areas of computer science, but also of considerable intrinsic mathematical and logical interest. Its defining feature is the addition of inductive definitions to modal logic; thereby it achieves a great increase in expressive power, and an equally great increase in difficulty of understanding. It includes many of the logics used in systems verification, and is quite straightforward to evaluate. It also provides one of the strongest examples of the connections between modal and temporal logics, automata theory and the theory of games.

In this chapter, we survey a range of the questions and results about the modal mu-calculus and related logics. For the most part, we remain at survey level, giving only outlines of proofs; but in places, determined partly by our own interests and partly by our sense of which problems have been — or had been — the longest-standing thorns in the side of the mu-calculus community, we go into more detail.

We start with an account of the historical context leading to the introduction of the modal mu-calculus. Then we define the logic formally, describe some approaches to gaining an intuitive understanding of formulae, and establish the main theorem about the semantics. Following that, we discuss how the modal mu-calculus has the tree model property and relates to some other temporal logics, to automata and to games. Next, an account of decidability is given — this is one of the thorns, at least for those who find automata prickly. We then consider briefly completeness, bisimulation invariance and then the concept of fixpoint alternation, which plays a part in several interesting questions about the logic. Finally, we look at some generalizations of the logic.

Before proceeding to the content of the chapter, we take this opportunity to thank Yde Venema and Johan van Benthem for extensive and helpful comments on drafts of this chapter.

Notation: $L\mu$ means the modal mu-calculus, considered as a logical language (not as a theory). In general, the notation follows as much as possible the standards for this book, but because $L\mu$ is mostly studied in a setting with rather different traditions, and because we also need to notate several other concepts, we have made some compromises. Few of these should cause any difficulty, but let us note the following. Since $\rightarrow$ is often used to represent the transition relation in models (alias the accessibility relation from modal logic), we use $\Rightarrow$ rather than $\rightarrow$ for boolean implication. Structures, frames and models for $L\mu$ are usually viewed as transition systems, and so are usually called $\mathfrak{T}$ with state space $\mathfrak{S}$. States within systems (i.e. worlds in the language of modal logic) are typically s, t, whereas p, q, r are states in an automaton. Hence we write atomic propositions with capital $P, Q, \ldots$ rather than $p, q, \ldots$, and similarly variables ranging over sets of states are written X, Y.

2 CONTEXTUAL BACKGROUND

The modal mu-calculus comes not from the philosophical tradition of modal logic, but from the application of modal and temporal logics to program verification. In this section, we outline the historical context for $L\mu$.

2.1 Modal logics in program verification

The application of modal and temporal logics to programs is part of a line of program verification going back to the 1960s and program schemes and Floyd–Hoare logic. Originally the emphasis was on *proof*: Floyd–Hoare logic allows one to make assertions about programs, and there is a proof system to verify these assertions. This line of work has, of course, continued and flourished, and today there are highly sophisticated theories for proving properties of programs, with equally sophisticated machine support for these theories. However, the use of proof systems has some disadvantages, and one hankers after a more purely algorithmic approach to simple problems. One technique was pioneered by Manna and Pnueli [48], who turned program properties into questions of satisfiability or validity in first order logic, which can then be attacked by means that are not just proof-theoretic; this idea was later applied by them to linear temporal logics.

During the 1970s, the theory of program correctness was extended by investigating more powerful logics, and studying them in a manner more similar to the traditions of mathematical logic. A family of logics which received much attention was that of dynamic logics, which can be seen as extending the ideas of Hoare logic [57]. Dynamic logics are modal logics, where the different modalities correspond to the execution of different programs — the formula $\langle\alpha\rangle\phi$ is read as 'it is possible for α to execute and result in a state satisfying ϕ'. The programs may be of any type of interest; the variety of dynamic logic most often referred to is a propositional language in which the programs are built from atomic programs by regular expression constructors; henceforth, Propositional Dynamic Logic, PDL, refers to this logic. PDL is interpreted with respect to a model on a Kripke structure, formalizing the notion of the global state in which programs execute and which they change — each point in the structure corresponds to a possible state, and programs determine a relation between states giving the changes effected by the programs.

Once one has the idea of a modal logic defined on a Kripke structure, it becomes quite natural to think of the finite case and write programs which just check whether a formula is satisfied. This idea was developed in the early 80s by Clarke, Emerson, Sistla and others. They worked with a logic that has much simpler modalities than PDL — in fact, it has just a single 'next state' modality — but which has built-in temporal connectives such as 'until'. This logic is CTL, and it and its extensions remain some of the most popular logics for expressing properties of systems.

Meanwhile, the theory of process calculi was being developed in the late 70s, most notably by Milner [50]. An essential component was the use of labelled Kripke structures ('labelled transition systems') as a raw model of concurrent behaviour. An important difference between the use of Kripke structures here and their use in program correctness was that the states are the behaviour expressions themselves, which model concurrent systems, and the labels on the accessibility relation (the transitions) are simple actions (and not programs). The criterion for behavioural equivalence of process expressions was defined in terms of observational equivalence (and later in terms of bisimulation relations). Hennessy and Milner introduced a primitive modal logic in which the modalities refer to actions: $\langle a\rangle\phi$ 'it is possible to do an a action and then have ϕ be true', and its dual $[a]\phi$ 'ϕ holds after every a action'. Together with the usual boolean connectives, this gives Hennessy–Milner logic [31], HML, which was introduced as an alternative exposition of observational equivalence. However, as a logic HML is obviously inadequate to express many properties, as it has no means of saying 'always in the future' or other temporal connectives — except by allowing infinitary conjunction. Using an infinitary