Abstract This chapter presents that part of the theory of the $\mu$-calculus that is relevant to the model-checking problem as broadly understood. The $\mu$-calculus is one of the most important logics in model checking. It is a logic with an exceptional balance between expressiveness and algorithmic properties.

The chapter describes at length the game characterization of the semantics of the $\mu$-calculus. It discusses the theory of the $\mu$-calculus starting with the tree-model property, and bisimulation invariance. Then it develops the notion of modal automaton: an automaton-based model behind the $\mu$-calculus. It gives a quite detailed explanation of the satisfiability algorithm, followed by results on alternation hierarchy, proof systems, and interpolation. Finally, the chapter discusses the relation of the $\mu$-calculus to monadic second-order logic as well as to some program and temporal logics. It also presents two extensions of the $\mu$-calculus that allow us to address issues such as inverse modalities.

26.1 Introduction

The $\mu$-calculus is one of the most important logics in model checking. It is a logic with an exceptional balance between expressiveness and algorithmic properties. In this chapter we present that part of the theory of the $\mu$-calculus that seems to us most relevant to the model-checking problem as broadly understood.

This chapter is divided into three parts. In Sect. 26.2 we introduce the logic, and present some basic notions such as: special forms of formulas, vectorial syntax, and alternation depth of fixpoints. The largest part of this section is concerned with a characterization of the semantics of the logic in terms of games. We give a relatively detailed exposition of the characterization, since in our opinion this is one of the central tools in the theory of the $\mu$-calculus. The section ends with an overview of approaches to the model-checking problem for the logic.

J. Bradfield
University of Edinburgh, Edinburgh, UK

I. Walukiewicz (✉)
CNRS, University of Bordeaux, Bordeaux, France
e-mail: igw@labri.fr
Section 26.3 goes deeper into the theory of the \( \mu \)-calculus. It starts with the tree-model property, and bisimulation invariance. Then it develops the notion of modal automaton: an automaton-based model behind the \( \mu \)-calculus. This model is then often used in the rest of the chapter. We continue the section with a quite detailed explanation of the satisfiability algorithm. This is followed by results on alternation hierarchy, proof systems, and interpolation. We finish with a division property that is useful for modular verification and synthesis.

Section 26.4 presents the \( \mu \)-calculus in a larger context. We relate the logic to monadic second-order logic as well as to some program and temporal logics. We also present two extensions of the \( \mu \)-calculus that allow us to express inverse modalities, some form of equality, or counting.

This chapter is short, given the material that we would like to cover. Instead of being exhaustive, we try to focus on concepts and ideas we consider important and interesting from the perspective of the model-checking problem as broadly understood. Since concepts often give more insight than enumeration of facts, we give quite complete arguments for the main results we present.

### 26.2 Basics

In this section we present some basic notions and tools of the theory of the \( \mu \)-calculus. We discuss some special forms of formulas such as guarded or vectorial forms. We introduce also the notion of alternation depth. Much of this section is devoted to a characterization of the semantics of the logic in terms of parity games, and its use in model checking. The section ends with an overview of model-checking methods and results.

#### 26.2.1 Syntax and Semantics

The \( \mu \)-calculus is a logic describing properties of transition systems: potentially infinite graphs with labeled edges and vertices. Often the edges are called transitions and the vertices states. Transitions are labeled with actions, \( \text{Act} = \{a, b, c, \ldots\} \), and the states with sets of propositions, \( \text{Prop} = \{p_1, p_2, \ldots\} \). Formally, a transition system is a tuple:

\[
\mathcal{M} = \langle S, \{R_a\}_{a \in \text{Act}}, \{P_i\}_{i \in \mathbb{N}} \rangle
\]

consisting of a set \( S \) of states, a binary relation \( R_a \subseteq S \times S \) defining transitions for every action \( a \in \text{Act} \), and a set \( P_i \subseteq S \) for every proposition. A pair \( (s, s') \in R_a \) is called an \( a \)-transition.

We require a countable set of variables, whose meanings will be sets of states. These can be bound by fixpoint operators to form fixpoint formulas. We use \( \text{Var} = \{X, Y, Z \ldots\} \) for variables.