The mu-calculus and Model Checking

Julian Bradfield, Igor Walukiewicz

The mu-calculus and model-checking

Julian Bradfield and Igor Walukiewicz

Abstract This chapter presents a part of the theory of the mu-calculus that is relevant to the, broadly understood, model-checking problem. 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 in 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 the results on alternation hierarchy, proof systems, and interpolation. Finally, the chapter discusses the relations 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.

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 a part of the theory of the mu-calculus that seems to us most relevant to the, broadly understood, model-checking problem.

Julian Bradfield
Laboratory for Foundations of Computer Science, University of Edinburgh,

Igor Walukiewicz
LaBRI, Bordeaux University

This chapter is divided into three parts. In Section 2 we introduce the logic, and present some basic notions such as: special forms of formulas, vectorial syntax, 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.

Section 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. It is followed by the results on alternation hierarchy, proof systems, and interpolation. We finish with a division property that is useful for modular verification and synthesis.

Section 4 presents the mu-calculus in the 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 broadly understood model-checking problem. Since concepts often give more insight than enumeration of facts, we give quite complete arguments for the main results we present.

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 like 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.

2.1 Syntax and semantics

The μ-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*, $Act = \{a, b, c, \dots\}$, and the states with sets of *propositions*, $Prop =$

$\{p_1, p_2, \dots\}$. Formally, a *transition system* is a tuple:

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

consisting of a set S of states, a binary relation $R_a \subseteq S \times S$ defining transitions for every action $a \in Act$, 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 $Var = \{X, Y, Z \dots\}$ for variables.

Syntax. The formulas of the logic are constructed using conjunction, disjunction, modalities, and fixpoint operators. The set of μ-calculus formulas, $\mathcal{F}_{mc}$ is the smallest set containing:

- p and $\neg p$ for all propositions $p \in Prop$;
- X for all variables $X \in Var$;
- $\alpha \vee \beta$ as well as $\alpha \wedge \beta$, if α, β are formulas in $\mathcal{F}_{mc}$;
- $\langle a \rangle \alpha$ and $[a]\alpha$, if $a \in Act$ is an action and α is a formula in $\mathcal{F}_{mc}$;
- $\mu X.\alpha$ and $\nu X.\alpha$, if $X \in Var$ is a variable and $\alpha \in \mathcal{F}_{mc}$ is a formula.

Disambiguating parentheses are added when necessary. It is generally agreed that $\langle a \rangle$ and $[a]$ bind more tightly than boolean operators, but opinions vary on whether μ and ν bind tighter or looser than booleans. We will assume that they bind looser. For example, consider the important formula 'infinitely often p on some path', which fully parenthesized is $\nu Y.\big(\mu X.((p \wedge \langle a \rangle Y) \vee \langle a \rangle X)\big)$. We shall write it as $\nu Y.\, \mu X.\, (p \wedge \langle a \rangle Y) \vee \langle a \rangle X$.

We write $\sigma X.\alpha$ to stand for $\mu X.\alpha$ or $\nu X.\alpha$. We will use tt as an abbreviation of $(p_1 \vee \neg p_1)$, and ff for $(p_1 \wedge \neg p_1)$. Note that there is no negation operation in the syntax; we just permit negations of propositions. Actually, the operation of the negation of a sentence will turn out to be definable.

Semantics. The semantics of the logic is concise and yet of intriguing depth. We will see later that it pays to study it in detail from different points of view. A meaning of a formula in a transition system is a set of states satisfying the formula. Since a formula may have free variables, their meaning should be fixed before evaluating the formula. As with formulas, the meaning of a variable will be a set of states of the transition system. More formally, given a transition system $\mathcal{M} = \langle S, \{R_a\}_{a \in Act}, \{P_i\}_{i \in \mathbb{N}} \rangle$ and a valuation $\mathcal{V} : Var \to \mathcal{P}(S)$ we define the meaning of a formula $[\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}}$ by induction on its structure. The meaning of variables is given by the valuation. The meanings of propositional constants and their negations are given by the transition system.

$$[\![X]\!]^{\mathcal{M}}_{\mathcal{V}} = \mathcal{V}(X), \quad \text{for every } X \in \mathcal{V};$$
$$[\![p_i]\!]^{\mathcal{M}}_{\mathcal{V}} = P_i \quad \text{and} \quad [\![\neg p_i]\!]^{\mathcal{M}}_{\mathcal{V}} = S - P_i, \quad \text{for every } p_i \in Prop.$$

Disjunction and conjunction are interpreted as the union and the intersection:

$$[\![\alpha \vee \beta]\!]^{\mathcal{M}}_{\mathcal{V}} = [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} \cup [\![\beta]\!]^{\mathcal{M}}_{\mathcal{V}} \qquad [\![\alpha \wedge \beta]\!]^{\mathcal{M}}_{\mathcal{V}} = [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} \cap [\![\beta]\!]^{\mathcal{M}}_{\mathcal{V}}.$$

The meaning of modalities is given by transitions. The formula $\langle a\rangle\alpha$ holds in some state if it has an outgoing a-transition to some state satisfying α. Dually, the formula $[a]\alpha$ holds in some state if all its outgoing a-transitions go to states satisfying α:

$$\begin{aligned}[\![\langle a\rangle\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} &= \{s \in S : \exists s'.R_a(s,s') \wedge s' \in [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}}\},\\ [\![[a]\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} &= \{s \in S : \forall s'.R_a(s,s') \Rightarrow s' \in [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}}\}.\end{aligned}$$

Finally, the μ and ν constructs are interpreted as fixpoints of operators on sets of formulas. A formula $\alpha(X)$ containing a free variable X can be seen as an operator on sets of states mapping a set S' to the semantics of α when X is interpreted as S', in symbols: $S' \mapsto [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}[S'/X]}$. Since by definition of the basic operators of the logic this operator is monotonic, it has well defined least and greatest fixpoints. Formally,

$$\begin{aligned}[\![\mu X.\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} &= \bigcap\{S' \subseteq S : [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}[S'/X]} \subseteq S'\},\\ [\![\nu X.\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} &= \bigcup\{S' \subseteq S : S' \subseteq [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}[S'/X]}\}.\end{aligned}$$

We will often write $\mathcal{M}, s, \mathcal{V} \vDash \alpha$ instead of $s \in [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}}$. Moreover we will omit $\mathcal{V}$ or $\mathcal{M}$ if it is not important, or clear from the context.

Examples: The simplest formulas are just those of modal logic: $\langle a\rangle tt$ means 'there is transition labeled by a'. With one fixpoint, we can talk about termination properties of paths in a transition system. The formula $\mu X.[a]X$ means that all sequences of a-transitions are finite. The formula $\nu Y.\langle a\rangle Y$ means that there is an infinite sequence of a-transitions. We can then add a predicate p, and obtain $\nu Y.\, p \wedge \langle a\rangle Y$ formula saying that there is an infinite sequence of a-transitions, and all states in this sequence satisfy p. The formula $\mu X.\langle a\rangle X$ is just false, but the formula $\mu X.\, p \vee \langle a\rangle X$ says that there is a sequence of a-transitions leading to a state where p holds. With two fixpoints, we can write fairness formulas, such as $\nu Y.\, \mu X.\, (p \wedge \langle a\rangle Y) \vee \langle a\rangle X$ meaning 'on some a-path there are infinitely many states where p holds'. Changing the order of fixpoints we get $\mu X.\, \nu Y.\, (p \wedge \langle a\rangle Y) \vee \langle a\rangle X$ saying 'on some a-path almost always p holds'. To see why these formulas mean what they do, one can of course use the semantics directly, but it is often easier to use some alternative approaches that we introduce in the following. As these examples suggest, the semantics depends on the order of fixpoint operators, and the expressive power increases with the number of fixpoints (cf. Section 2.2).

Some syntactic conventions. The fixpoint operators μX and νX bind occurrences of the variable X, in the sense that the meaning of $\mu X.\alpha$ does not depend on the valuation of X. We leave to the reader the formal definition

of bound and free occurrence of a variable in a formula. A *sentence* is a formula without free variables. In particular, the meaning of a sentence does not depend on the valuation of variables. By $\alpha[\beta/X]$ we denote the result of substitution of β for every free occurrence of X; when doing this we suppose that the free variables of β are disjoint from the bound variables of α. Clearly $\mu X.\alpha$ is equivalent to $\mu Y.(\alpha[Y/X])$, so we can always make sure that no variable has at the same time a free and a bound occurrence in a formula.

In order to underline the dependency of the value of α on X, we will often write $\mu X.\alpha(X)$ instead of $\mu X.\alpha$. In this context we write $\alpha(\beta)$ for $\alpha[\beta/X]$. We immediately employ this notation to introduce the idea of unfolding. A fixpoint formula $\mu X.\alpha(X)$ is equivalent to its *unfolding*, $\alpha(\mu X.\alpha(X))$. This is a very useful rule that allows us to "delay" reasoning about fixpoints. The equivalence of a formula with its unfolding follows directly from the fact that μ is a fixpoint operator. Of course the same applies for ν in place of μ.

Semantics based on approximations. There is another very convenient way of defining meanings of fixpoint constructs. It comes directly from the Knaster–Tarski theorem characterizing fixpoints in a complete lattice in terms of their approximations. Let us start with a definition of formal approximations of fixpoint formulas: $\mu^\tau X.\alpha(X)$ and $\nu^\tau X.\alpha(X)$ for every ordinal τ. The meaning of $\mu^0 X.\alpha(X)$ is the empty set. The meaning of $\mu^{\tau+1} X.\alpha(X)$ is that of $\alpha(Z)$ where Z is interpreted as $\mu^\tau X.\alpha(X)$. Finally, the meaning of $\mu^\tau X.\alpha(X)$ when τ is a limit ordinal is the least upper bound of meanings of $\mu^\rho X.\alpha(X)$ for $\rho < \tau$. Similarly for $\nu^\tau X.\alpha(X)$ but for the fact that $\nu^0 X.\alpha(X)$ is the set of all states, and the greatest lower bound is taken when τ is a limit ordinal.

Example: Let us look at the meaning of approximations of a formula $\mu X.[a]X$:

$$
\begin{array}{lll}
\mu^0 X.[a]X = \varnothing & & \text{false} \\
\mu^1 X.[a]X = [a]\varnothing & & \text{states with no } a\text{-path} \\
\mu^2 X.[a]X = [a][a]\varnothing & & \text{states with no } aa\text{-path} \\
\quad \ldots & & \\
\mu^\omega X.[a]X = \bigcup_{n<\omega} \mu^n & & \text{states s.t. } \exists n.\ \text{no } a^n\text{-path} \\
\quad \ldots & &
\end{array}
$$

If every state has only finitely many a-successors, then the approximation *closes at* ω, i.e. $\mu^{\omega+1} = \mu^\omega$; but for infinite-branching systems, we may need to go further, and the approximation closes at the least upper bound of ordinal heights of an a-tree in the system (cf. Figure 5, page 17). In general, the least ordinal such that $\mu^\tau X.\alpha = \mu^{\tau+1} X.\alpha$ in a transition system $\mathcal{M}$ is called the closure ordinal of $\mu X.\alpha$ in $\mathcal{M}$. The closure ordinal always exists; its cardinal is bounded by the cardinality of the transition system.

For a more complex example, consider the formula $\nu Y.\, \mu X.\, \langle a\rangle((p\wedge Y)\vee X)$ which is another way of writing 'along some a-path there are infinitely many states where p holds' formula we have seen in the example on page 4. Here we have to calculate the approximations of the ν formula, and during each such calculation, we have to calculate the approximations of the μ formula, *relative to the current ν approximation*. For ease of tabulation, write ν^τ for $\nu^\tau Y.\, \mu X.\, \langle a\rangle((p \wedge Y) \vee X)$, and $\mu^{\tau,\tau'}$ for $\mu^{\tau'} X.(\langle a\rangle((p \wedge Y) \vee X))[\nu^\tau/Y]$. Now we have, with some abuse of notation:

$$
\begin{array}{lll}
\nu^0 & S & \\
\quad \mu^{0,0} & \quad \varnothing & \\
\quad \mu^{0,1} & \quad [\![\langle a\rangle((p \wedge S) \vee \mu^{0,0})]\!] = [\![\langle a\rangle p]\!] & \\
\quad \mu^{0,2} & \quad [\![\langle a\rangle((p \wedge S) \vee \mu^{0,1})]\!] = [\![\langle a\rangle(p \vee \langle a\rangle p)]\!] & \\
\quad \ldots & & \\
\nu^1 = \mu^{0,\infty} & \langle a\rangle\text{eventually}(p) & \\
\quad \mu^{1,1} & \quad [\![\langle a\rangle((p \wedge \nu^1) \vee \varnothing)]\!] = [\![\langle a\rangle(p \wedge \nu^1)]\!] & \\
\quad \mu^{1,2} & \quad [\![\langle a\rangle((p \wedge \nu^1) \vee \mu^{1,1})]\!] = [\![\langle a\rangle\big((p \wedge \nu^1) \vee \langle a\rangle(p \wedge \nu^1)\big)]\!] & \\
\quad \ldots & \ldots & \\
\nu^2 = \mu^{1,\infty} & \text{eventually}(p \wedge \langle a\rangle\text{eventually}(p)) & \\
\quad \ldots & \ldots & \\
\nu^\infty & \text{infinitely often } p &
\end{array}
$$

In this example, 'eventually p' means 'on some a-path, p will occur'. If the modality $\langle a\rangle$ were replaced by the $[a]$ modality, then it would mean 'on every a-path, p will occur'.

Negation. Since the syntax we propose does not have the negation operation, it is useful to see that negation can be defined in the language. We first define by induction on the structure a formal negation operation $\neg\alpha$ on formulas and then state that it has the required properties.

$$
\begin{array}{cc}
\neg(\neg p) = p & \neg(\neg X) = X \\
\neg(\alpha \vee \beta) = \neg\alpha \wedge \neg\beta & \neg(\alpha \wedge \beta) = \neg\alpha \vee \neg\beta \\
\neg\langle a\rangle\alpha = [a]\neg\alpha & \neg[a]\alpha = \langle a\rangle\neg\alpha \\
\neg\mu X.\alpha(X) = \nu X.\neg\alpha(\neg X) & \neg\nu X.\alpha(X) = \mu X.\neg\alpha(\neg X)
\end{array}
$$

Observe that when applying this translation to a formula without free variables, the final result has all variables occurring un-negated, because of the two negations introduced when negating fixpoint expressions.

Fact 1 (Definability of negation) *For every sentence α, every transition system $\mathcal{M}$ over the set of states S, and every valuation $\mathcal{V}$:*

$$[\![\neg\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} = S - [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} \ .$$

Examples: The negation of the 'everywhere always p' formula $\nu X.\, p \wedge [a]X$ is $\neg\nu X.\, p \wedge [a]X = \mu X.\neg(p \wedge [a](\neg X)) = \mu X.\, \neg p \vee \neg[a](\neg X) = \mu X.\, \neg p \vee \langle a\rangle X$, the 'eventually somewhere $\neg p$' formula.

For a more complicated example let us come back to $\nu Y.\, \mu X.\, (p \wedge \langle a\rangle Y) \vee \langle a\rangle X$ meaning 'along some a-path there are infinitely many states where p holds'. Its negation is $\mu Y.\, \nu X.\, (\neg p \vee [a]Y) \wedge [a]X$ expressing, in a slightly cryptic way, 'on every path almost always $\neg p$'.

Special forms of formulas. Let us mention some useful special forms for formulas. First, as we have noted above, we can require that bound and free variables are different. We can also require that every variable is bound at most once in a formula. If both of these are the case, we say the formula is *well-named*. Moreover, we can even ensure that in every formula $\mu X.\alpha(X)$ variable X appears only once in $\alpha(X)$. This is because $\mu X.\mu Y.\alpha(X,Y)$ is equivalent to $\mu X.\alpha(X,X)$. Similarly for $\nu X.\alpha(X)$.

Another useful syntactic property is *guardedness*. A variable Y is guarded in $\beta(Y)$ if all occurrences of Y are preceded (not necessary directly) by a modality. For example, Y is guarded in $\langle a\rangle\mu X.(X \wedge Y \wedge p)$. A formula is *guarded* if for every subformula $\sigma Y.\beta(Y)$, variable Y is guarded in $\beta(Y)$. It turns out that every formula is equivalent to a guarded formula.

The algorithm for constructing an equivalent guarded formula uses an operation of removing open occurrences of a variable. An occurrence of a variable is *open* if it is neither guarded, nor preceded by a fixpoint operator. To see how it works, consider a formula $\mu Y.\beta(Y)$ and suppose we want to obtain an equivalent formula without open occurrences of Y in $\beta(Y)$. For this it suffices to replace every open occurrence of Y in $\beta(Y)$ by ff. To see why this may work, observe that $\mu Y.\, Y \wedge \gamma(Y)$ is equivalent to ff while $\mu Y.\, Y \vee \gamma(Y)$ is equivalent to $\mu Y.\gamma(Y)$. Now, due to laws of propositional logic, the formula $\beta(Y)$ is equivalent to $Y \wedge \gamma(Y)$ or $Y \vee \gamma(Y)$ for some $\gamma(Y)$ with no open occurrence of Y. We get that $\mu Y.\beta(Y)$ is equivalent to $\mu Y.Y \wedge \gamma(Y)$ or $\mu Y.Y \vee \gamma(Y)$. By the observation above, these in turn are equivalent to ff or to $\mu Y.\gamma(Y)$, respectively. The translation for $\nu Y.\beta(Y)$ is dual: open occurrences of Y are replaced by tt.

To convert a formula to a guarded formula we repeatedly remove open occurrences of variables starting from innermost fixpoint formulas. For example, consider a formula $\nu Y.\mu X.\alpha(X,Y)$, where $\alpha(X,Y)$ does not have fixpoint subformulas. We first remove open occurrences of X in $\mu X.\alpha(X,Y)$. In the obtained formula, $\mu X.\alpha'(X,Y)$, all occurrences of X are guarded as the formula does not have proper fixpoint subformulas. In consequence, every occurrence of Y in $\alpha'(\mu X.\alpha(X,Y),Y)$ is either open or guarded. Hence we

remove open occurrences of Y in $\nu Y.\alpha'(\mu X.\alpha(X,Y),Y)$ and obtain a guarded formula.

Fact 2 (Special form of formulas) *Every formula can be transformed to an equivalent guarded, well-named formula. Moreover, one can require that in every subformula of the form $\sigma X.\beta(X)$ variable X appears at most once in $\beta(X)$.*

As observed in [54], contrary to some claims in the literature, the transformation to a guarded form described above can induce an exponential growth in the size of the formula. It is not known if there is a better transformation. Often it is enough to remove open occurrences of bound variables though, and this transformation does not increase the size of the formula. A way of avoiding exponential blowup is to use vectorial syntax described below [109].

Vectorial syntax. The original syntax of the mu-calculus allows us to freely mix all types of operators. In some contexts it is more interesting to have a formula in a prenex form where all the fixpoint operators are on the outside. This is possible in a vectorial syntax we will now introduce. Another advantage of this syntax is that it is in general more compact, as it allows the sharing of common subformulas.

A *modal formula* is a formula of the mu-calculus without fixpoint operators. As we do not allow negation of a variable in the syntax, this formula is positive. A sequence $\boldsymbol{\alpha} = (\alpha^1, \ldots, \alpha^n)$ of n modal formulas is a *vectorial mu-calculus formula of height n*. If $\mathbf{X} = X^1, \ldots, X^n$ is a sequence of n variables and $\boldsymbol{\alpha}$ a vectorial formula of height n then $\mu\mathbf{X}.\boldsymbol{\alpha}$ and $\nu\mathbf{X}.\boldsymbol{\alpha}$ are vectorial formulas of height n.

The meaning of a vectorial formula of height n is an n-tuple of sets of states. Apart from that, the semantics is analogous to the scalar (i.e., ordinary) mu-calculus. More precisely, if $\boldsymbol{\alpha} = (\alpha^1, \ldots, \alpha^n)$ is a sequence of modal formulas then its meaning in a model $\mathcal{M}$ with a valuation $\mathcal{V}$ is $[\![\alpha^1]\!]^{\mathcal{M}}_{\mathcal{V}} \times \cdots \times [\![\alpha^n]\!]^{\mathcal{M}}_{\mathcal{V}}$. Observe that with the variables $\mathbf{X}$ distinguished, the meaning of $\boldsymbol{\alpha}$ is a function from $\mathcal{P}(S)^n$ to $\mathcal{P}(S)^n$. The meaning of $\mu\mathbf{X}.\boldsymbol{\alpha}$ is then the least fixed-point of this function. Similarly for $\nu\mathbf{X}.\boldsymbol{\alpha}$.

It turns out that vectorial and scalar mu-calculi have the same expressive power. This is one more example of remarkable closure properties of the logic. The translation from scalar to vectorial formulas is rather direct. One introduces a variable for every subformula and then writes a fixpoint formula in the obvious way. The obtained vectorial formula has the property that the first component of its meaning is exactly the meaning of the scalar formula. The translation in the other direction relies on a repeated use of the so-called Bekič principle [10]:

$$\mu\begin{bmatrix}X\\Y\end{bmatrix}.\begin{bmatrix}\alpha(X,Y)\\\beta(X,Y)\end{bmatrix} = \begin{bmatrix}\mu X.\,\alpha(X,\mu Y.\beta(X,Y))\\\mu Y.\,\beta(\mu X.\alpha(X,Y),Y)\end{bmatrix}.$$

This principle allows us to eliminate the prefix of fixpoint operators. In the result we obtain a vector of formulas. The formula at the i-th coordinate will give the semantics of the i-th coordinate of the original vectorial formula.

Fact 3 (Vectorial syntax) *Every μ-calculus formula can be converted to an equivalent vectorial formula. Every vectorial formula can be converted to an equivalent (scalar) μ-calculus formula.*

The translation from scalar to vectorial form does not yield a blow-up in size. It is conjectured that vectorial formulas may be exponentially smaller than their scalar equivalents.

2.2 Alternation depth

The examples above suggest that the power for the logic comes from fixpoint operators. While most useful properties can be expressed with few fixpoints, it is the nesting of the two types of fixpoints that is the source of both expressive power and algorithmic difficulties. We introduce some notions to state this formally.

Let α be a well-named formula. So for every bound variable Y we have a unique subformula $\sigma Y.\beta_Y$ in α; and it makes sense to say that Y is a μ-variable or a ν-variable depending on the binder. This syntactic convention makes it easier to define the notion of alternation depth, otherwise we would need to refer to specific occurrences of variables.

Definition 1 (Alternation depth). The *dependency order* on bound variables of α is the smallest partial order such that $X \leq_\alpha Y$ if X occurs free in $\sigma Y.\beta_Y$. The *alternation depth of a μ-variable* X in formula α is the maximal length of a chain $X_1 \leq_\alpha \cdots \leq_\alpha X_n$ where $X = X_1$, variables $X_1, X_3, \ldots$ are μ-variables and variables $X_2, X_4, \ldots$ are ν-variables. Alternation depth of a ν-variable is defined similarly. *Alternation depth of formula* α, denoted $adepth(\alpha)$, is the maximum of alternation depths of variables bound in α, or zero if there are no fixpoints.

Examples: The now-familiar 'on some a-path there are infinitely many states where p holds' formula $\nu Y.\, \mu X.\, (p \wedge \langle a \rangle Y) \vee \langle a \rangle X$ is a canonical example of an alternation depth 2 formula since Y has alternation depth 2. Indeed $Y \geq X$ in the dependence order, and Y is a ν-variable while X is a μ-variable. In contrast, the 'there is a path where p holds almost always' formula $\mu X.\, \big(\nu Y.\, (p \wedge \langle a \rangle Y)\big) \vee \langle a \rangle X$ has alternation depth 1, since X does not occur free in $\nu Y.\, (p \wedge \langle a \rangle Y)$ and in consequence has alternation depth 1.

The following fact gives a first good reason for this seemingly complicated definition

Fact 4 (Alternation depth and unfolding) *A formula $\mu X.\beta(X)$ has the same alternation depth as its unfolding $\beta(\mu X.\beta(X))$. Similarly for the greatest fixpoint.*

Indeed, after renaming bound variables to avoid their repeated use, the dependency order of a sentence $\beta(\mu X.\beta(X))$ is a disjoint union of the dependency order for $\mu X.\beta(X)$ and that for $\beta(X)$. The number of alternations in the latter is not greater than in the former.

Alternation depth is a parameter that appears in many contexts. It is crucial in translations between the logic and automata. It induces a hierarchy with respect to expressive power. The complexity of all known model-checking algorithms depends exponentially on this parameter.

We will see alternation depth often in this chapter. At the moment let us only observe that this apparently technical definition becomes much more readable in vectorial syntax: the alternation depth is just the number of alternations between μ and ν in the prefix. It is tiresome but not difficult to check that the translations between scalar and vectorial formulas introduced on page 8 preserve alternation depth.

There is a commonly seen alternative formulation, analogous to the definition of arithmetic hierarchy. Rather than talking about the 'alternation depth', we may classify formulas into a hierarchy of Σ^μ_n and Π^μ_n classes according to the nesting of fixpoint operators. So, for example, Σ^μ_1 consists of formulas with only μ fixpoints, and Π^μ_1 consists of formulas having only ν fixpoints. Then Σ^μ_2 is the closure of Π^μ_1 under boolean operations, substitutions, and μ. Observe that unlike for arithmetic, we need to explicitly mention substitutions in the definition. Class Π^μ_2 is defined similarly but using closure under ν. It can be shown that a formula has alternation depth n if and only if it is syntactically both in Σ^μ_{n+1} and in Π^μ_{n+1}.

Examples: The 'always on every a-path p' formula $\nu X.p \wedge [a]X$ is a Π^μ_1 formula, and the 'on every a-path eventually p' formula $\mu X.p \vee [a]X$ is Σ^μ_1; both are alternation depth 1. However, 'on some a-path p holds almost always' formula $\mu X.\big(\nu Y.(p \wedge \langle a\rangle Y)\big) \vee \langle a\rangle X$ has also alternation depth 1 but it is neither Π^μ_1 nor Σ^μ_1. It is Σ^μ_2 because it can be obtained by substituting the (Π^μ_1 and therefore) Σ^μ_2 formula $\nu Y.(p \wedge \langle a\rangle Y)$ for Z in the (Σ^μ_1 and therefore) Σ^μ_2 formula $\mu X.Z \vee \langle a\rangle X$. It is also Π^μ_2, for the same reason. Note also that the alternation depth 2 formula 'on some a-path there are infinitely many states where p holds' written $\nu Y.\mu X.(p \wedge \langle a\rangle Y) \vee \langle a\rangle X$ is Π^μ_2 but *not* Σ^μ_2.

2.3 Semantics in terms of games

There are two good reasons why the mu-calculus has the right to its own chapter in this Handbook: expressive power and algorithmic properties. Indeed the logic can encode most of the other logics used in verification, and

still algorithmically it is not substantially more difficult than the others. Nevertheless, the mu-calculus has been relatively slow in gaining acceptance, mainly because of its compact syntax. It is quite difficult to decode the meaning of a formula using the semantic clauses presented above. This is why the semantics in terms of games that we introduce here is conceptually very useful.

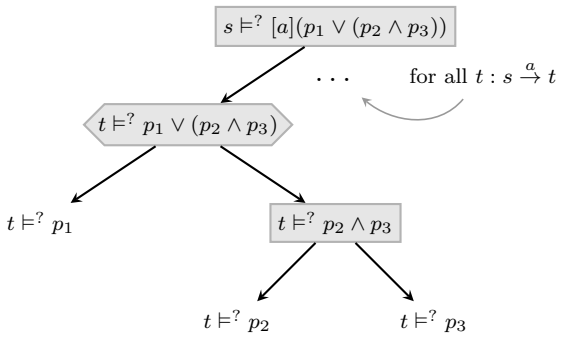

Fig. 1 Game for verifying $s \vDash^? [a](p_1 \vee (p_2 \wedge p_3))$.

To see what we are aiming at, consider a formula $[a](p_1 \vee (p_2 \wedge p_3))$. Suppose that we want to verify that the formula holds in a state s of some transition system $\mathcal{M}$. We describe the verification process as a game between two players: Eve and Adam. The goal of Eve will be to show that the formula holds, while Adam aims at the opposite.

The game is presented in Figure 1. The positions of Eve are pointed, and those of Adam are square. For example, the initial position belongs to Adam, and he has to choose there a state t reachable from s on an a-transition. The leaf position $t \vDash^? p_i$ is winning for Eve iff p_i holds in t. Looking at the game, it should be clear that the initial formula holds iff Eve has a strategy to reach a winning leaf. Her strategy is to choose in every position $t \vDash^? p_1 \vee (p_2 \wedge p_3)$ a disjunct that holds in t, if there is one.

To see a more challenging case consider the formula "infinitely often p on every path" $\nu Y.\mu Z.[a](Z \vee (p \wedge Y))$. Observe that apart from the two fixpoints the formula resembles very much the previous one. The game presented in Figure 2 is also similar. For fixpoints we just apply the unfolding rule; we use α_Y to stand for the whole formula and α_Z for its subformula $\mu Z.[a](Z \vee (p \wedge Y))$. We have not marked to whom belong nodes with fixpoint formulas since it is not important: they always have a unique successor. Observe that this time the game may be infinite: from the bottom rightmost node we restart the whole game; from the bottom leftmost node we restart from the μ-subformula. The main point is to be able to decide who is the winner of an infinite play. As it will turn out, we cannot just say that all infinite plays are winning for one of the players. In this example, we will declare a path

winning for Eve if the path passes infinitely often through the ν-formula (as in the rightmost bottom node).

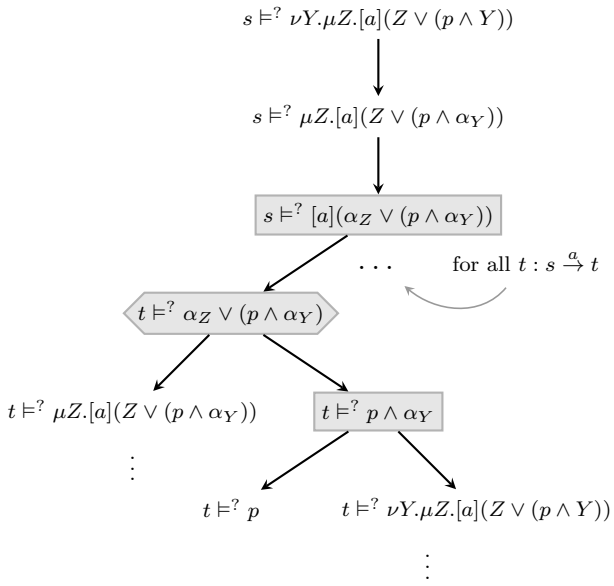

Fig. 2 Game for verifying $s \vDash^? \nu Y.\mu Z.[a](Z \vee (p \wedge Y))$.

In the following we will introduce the notion of a game with parity winning conditions, sufficient to deal with the mu-calculus. Parity conditions allow us to express properties like passing infinitely often through some node. Another chapter [20] of this Handbook describes many more variants of games used in model-checking. After presenting parity games, we will examine more closely the reduction of the model-checking problem to the problem of deciding the winner in a parity game constructed along the lines presented above.

2.3.1 Games

A *game* is a graph with a partition of nodes between two players, called Eve and Adam, and a set defining the winning condition. Formally it is a tuple

$$\mathcal{G} = \langle V, V_E, V_A, T \subseteq V \times V, Acc \subseteq V^\omega \rangle$$

where (V_E, V_A) is a partition of the set of nodes or *positions* V into those of Eve and those of Adam, T is the transition relation determining what are possible *successors* for each node, and Acc is a set defining the *winning condition*.

A *play* between Eve and Adam from some position $v \in V = V_E \cup V_A$ proceeds as follows: if $v \in V_E$ then Eve makes a choice of a successor, otherwise Adam chooses a successor; from this successor the same rule applies and the play goes on forever unless one of the parties cannot make a move. The player who cannot make a move loses. The result of an infinite play is an infinite path $v_0 v_1 v_2 \dots$ This *path is winning* for Eve if it belongs to *Acc*. Otherwise Adam is the winner.

In the game presented in Figure 3 the positions of Adam are marked with squares and the positions of Eve with diamonds. Additionally, each position is given a numerical *rank* in order to define *Acc* as we will see below. Observe that the unique position with no successors belongs to Adam, so he loses there. Let us say that Eve wins a play if it passes infinitely often through the position labeled with 2. For instance, if in the unique node for Eve she always chooses to go down, then she wins as 2 is on the loop. Actually Eve can also allow herself to go up, as then Adam has to go back to the position of rank 1. So as long as Eve goes infinitely often down she sees 2 infinitely often and wins.

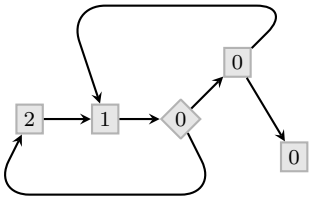

Fig. 3 A parity game.

A *strategy* for Eve is a function θ assigning to every sequence of nodes $\mathbf{v}$ ending in a node v from V_E a node $\theta(\mathbf{v})$ which is a successor of v. A *play respecting* θ is a sequence $v_0 v_1 \dots$ such that $v_{i+1} = \theta(v_0 \dots v_i)$ for all i with $v_i \in V_E$. The *strategy* θ *is winning for Eve* from a node v iff all the plays starting in v and respecting θ are winning. A *node is winning* if there exists a strategy winning from it. The strategies for Adam are defined similarly. A strategy is *positional* if it depends only on the last node in the sequence. So such a strategy can be represented as a function $\theta : V_E \to V$ and identified with a choice of edges in the graph of the game.

In this chapter we will consider only *parity winning conditions*. Such a condition is determined by a function $\Omega : V \to \{0, \dots, d\}$ in the following way:

$$Acc = \{v_0 v_1 \dots \in V^\omega : \limsup_{i \to \infty} \Omega(v_i) \text{ is even}\} .$$

Hence, each position is assigned a natural number, called *rank*, and we require that the largest among ranks appearing infinitely often is even. This condition, discovered by Mostowski [88] and independently by Emerson and Jutla [46], is the most useful condition in the context of the mu-calculus. The

condition "infinitely often 2, or finitely often both 2 and 1" from the game in Figure 3 is an example of a parity condition.

The main algorithmic question about such games is to decide who of the two players has a winning strategy from a given position. In other words to decide whether a given position is *winning for Eve* or *for Adam*. Principal results that we need about parity games are summarized in the following theorem. We refer the reader to [115, 20] for more details. We discuss the complexity issues at the end of Section 2.4.

Theorem 5 (Solving parity games [81, 46, 89]). *Every position of a game with a parity winning condition is winning for one of the two players. Moreover, a player has a positional strategy winning from each of his winning positions. It is algorithmically decidable who is a winner from a given position in a finite game with a parity condition.*

2.3.2 Verification game

We want to understand when a μ-calculus sentence α holds in a state s of a transition system $\mathcal{M}$. We characterize this by existence of a winning strategy in a specially constructed game $\mathcal{G}(\mathcal{M}, \alpha)$. More precisely, we want that $\mathcal{M}, s \vDash \alpha$ iff Eve has a winning strategy from a position corresponding to s and α in $\mathcal{G}(\mathcal{M}, \alpha)$. As we will need such a game for formulas with free variables as well, we will also take into account valuations. So, we will define a game $\mathcal{G}_{\mathcal{V}}(\mathcal{M}, \alpha)$, with $\mathcal{G}(\mathcal{M}, \alpha)$ being the special case when α is a sentence.

Positions in the game $\mathcal{G}_{\mathcal{V}}(\mathcal{M}, \alpha)$ are of the form (s, β) where s is a state of $\mathcal{M}$, and β is a formula from the *closure* of α, that is the smallest set containing α and closed under subformulas and unfolding. The intention is that Eve has a winning strategy from (s, β) iff $\mathcal{M}, s, \mathcal{V} \vDash \beta$.

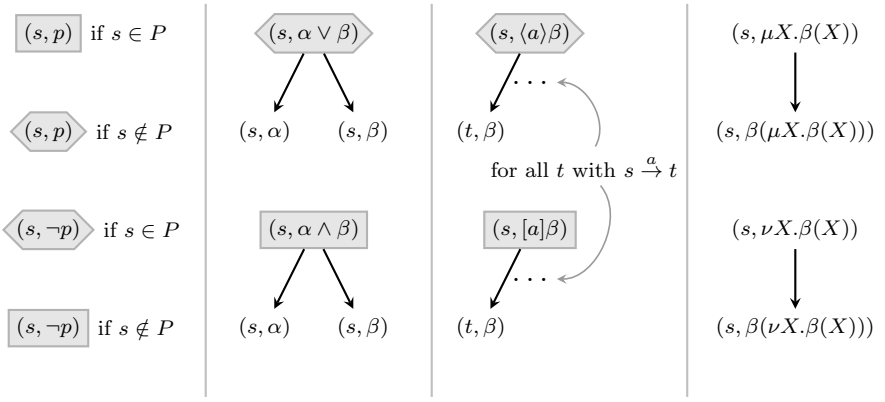

Fig. 4 Rules of the verification game $\mathcal{G}(\mathcal{M}, \alpha)$.

We define the rules of the game by induction on the syntax of the formula (cf. Figure 4). Clearly (s, p) should be declared winning for Eve if and only if proposition p holds in s. So we put no transition from this state and make it belong to Adam iff p holds in s. For a position $(s, \neg p)$ we proceed in the same way but exchange the roles of Adam and Eve. Observe that since there are no outgoing transitions, the player to whom the position belongs loses in it. For similar reasons a position (s, X), with X a variable, has no outgoing transitions and it belongs to Adam iff $s \in \mathcal{V}(X)$.

From positions $(s, \alpha \vee \beta)$ and $(s, \alpha \wedge \beta)$ we put transitions to (s, α) and to (s, β). Position $(s, \alpha \vee \beta)$ should belong to Eve, as $\alpha \vee \beta$ is satisfied in a state if and only if at least one of α or β is. This means that Eve has to express her opinion on which of the two formulas holds. Dually, $(s, \alpha \wedge \beta)$ belongs to Adam.

From positions $(s, \langle a \rangle \beta)$ and $(s, [a]\beta)$ there are transitions to (t, β) for all t reachable from s by an a-transition, that is for all t such that $(s, t) \in R_a$. Position $(s, \langle a \rangle \beta)$ should belong to Eve as in order for a formula to be satisfied there should be an a-edge to t satisfying β. Dually, $(s, [a]\beta)$ belongs to Adam.

Finally, from positions $(s, \mu X.\beta(X))$ and $(s, \nu X.\beta(X))$ there are transitions to $(s, \beta(\mu X.\beta(X)))$ and $(s, \beta(\nu X.\beta(X)))$ respectively. This corresponds to the intuition that a fixpoint is equivalent to its unfolding. As these positions have exactly one successor, it does not matter to which player they belong.

It remains to assign ranks to positions. We will be interested only in formulas starting with μ or ν; that is of the form $\mu X.\beta(X)$ or $\nu X.\beta(X)$. For a position with a formula of this form, we assign a rank in such a way that those starting with μ have odd ranks and those starting with ν have even ranks. Moreover, if γ is a subformula of β we require that the rank of γ is not bigger than that of β. One way to assign the ranks like this is to use the alternation depth (cf. Definition 1)

$$\begin{aligned}
\Omega(\gamma) &= 2 \cdot \lfloor adepth(X)/2 \rfloor && \text{if } \gamma \text{ is of the form } \nu X.\gamma'(X) \\
\Omega(\gamma) &= 2 \cdot \lfloor adepth(X)/2 \rfloor + 1 && \text{if } \gamma \text{ is of the form } \mu X.\gamma'(X) \\
\Omega(\gamma) &= 0 && \text{otherwise}
\end{aligned}$$

Observe that we are using alternation depth of X and not that of γ in this definition. This is because γ may contain formulas of big alternation depth not related with X. The sketch of the proof presented below provides more intuitions behind this definition of rank.

Examples: Consider formulas from the example on page 10. The rank of formula $\nu X.p \wedge [a]X$ is 0, while the rank of $\mu X.p \vee [a]X$ is 1. The rank of $\mu X.\big(\nu Y.(p \wedge \langle a \rangle Y)\big) \vee \langle a \rangle X$ is also 1 since the alternation depth of X is 1. But the rank of $\nu Y.\mu X.(p \wedge \langle a \rangle Y) \vee \langle a \rangle X$ is 2.

Having defined $\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)$, and $\mathcal{G}(\mathcal{M},\alpha)$ that is a special case when α is a sentence, we are ready to formulate one of the main theorems of this chapter.

Theorem 6 (Reducing model-checking to parity games [46]). *For every sentence α, transition system $\mathcal{M}$, and its state s: $\mathcal{M}, s \vDash \alpha$ iff Eve has a winning strategy from the position (s,α) in $\mathcal{G}(\mathcal{M},\alpha)$.*

The rest of this subsection is devoted to the proof of this theorem.

We fix $\mathcal{M}$ and α. For a valuation $\mathcal{V}$ we will denote by $\mathcal{G}_{\mathcal{V}}$ the game $\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)$. By induction on the structure of the formula we show that for every state s and valuation $\mathcal{V}$:

$$\mathcal{M}, s, \mathcal{V} \vDash \beta \quad \text{iff} \quad \text{Eve wins from } (s,\beta) \text{ in } \mathcal{G}_{\mathcal{V}} \ .$$

The cases when β is a proposition or a variable follow directly from the definition. Among the other cases, the interesting ones are for the fixpoint operators. We will do the one for μ. In the proof we do not assume that $\mathcal{M}$ is finite, and this is why we need to consider ordinals.

Take a formula $\mu X.\beta(X)$ and consider left to right implication. Using the characterization of μ via approximations, we know that there is an ordinal τ such that $\mathcal{M}, s, \mathcal{V} \vDash \mu^{\tau} X.\beta(X)$. We can suppose that τ is the smallest such ordinal. Directly from definitions of approximations (cf. page 5), it then follows that τ is a successor ordinal, so $\tau = \rho + 1$. In other words: $\mathcal{M}, s, \mathcal{V}_{\rho} \vDash \beta(X)$, where $\mathcal{V}_{\rho}$ is $\mathcal{V}$ modified so that the value of X is $[\![\mu^{\rho} X.\beta(X)]\!]^{\mathcal{M}}_{\mathcal{V}}$. We will do additional induction on ρ.

The outermost induction on the structure of the formula gives a winning strategy for Eve from $(s,\beta(X))$ in the game $\mathcal{G}_{\mathcal{V}_{\rho}}$. This strategy may reach a winning position (s',X) for some $s' \in \mathcal{V}_{\rho}(X)$. Since $\mathcal{V}_{\rho}(X) = [\![\mu^{\rho} X.\beta(X)]\!]^{\mathcal{M}}_{\mathcal{V}}$, the induction assumption on ρ tells us that Eve has a winning strategy from $(s',\mu X.\beta(X))$ in $\mathcal{G}_{\mathcal{V}}$. So the winning strategy for Eve in $\mathcal{G}_{\mathcal{V}}$ from $(s,\mu X.\beta(X))$ is to go to $(s,\beta(\mu X.\beta(X))$ and then follow the winning strategy from $(s,\beta(X))$ in the game $\mathcal{G}_{\mathcal{V}_{\rho}}$. If a play respecting this strategy reaches $(s',\mu X.\beta(X))$, Eve can change to a winning strategy that exists there.

For the implication in the other direction, we suppose that Eve has a winning strategy from $(s,\mu X.\beta(X))$ in $\mathcal{G}_{\mathcal{V}}$. Note that this position has an odd rank, say r. The first crucial observation is a refinement of Fact 4. For every subformula $\sigma Y.\gamma(Y)$ of $\beta(\mu X.\beta(X))$ there are two possibilities: (i) either $\mu X.\beta(X)$ is not a subformula of $\gamma(Y)$, or (ii) $\mathit{adepth}(Y) \leq \mathit{adepth}(X)$ and the inequality is strict if Y is a ν-variable. Indeed suppose that $\mu X.\beta(X)$ is a subformula of $\sigma Y.\gamma(Y)$ that is itself a subformula of $\beta(\mu X.\beta(X))$. Then $\mu X.\beta(X)$ can be written as $\mu X.\theta(X,\sigma Y.\gamma'(X,Y))$, and the required inequalities between the alternation depths of X and Y follow by the definition. This observation would not be true if we have defined the rank of a position using the alternation depth of $\mu X.\beta(X)$ instead of the alternation depth of the variable X. For example, when $\mu X.\beta(X)$ has a subformula $\nu Y.\gamma(Y)$

that itself has a subformula with an alternation depth bigger than that of $\mu X.\beta(X)$.

The observation from the preceding paragraph implies that if a game from a position $(s, \mu X.\beta(X))$ reaches some position (s', β') of a rank bigger than r then $\mu X.\beta(X)$ is not a subformula of β'. As the rank r is odd and every play is winning, this means that on every play there are finitely many positions of the form $(s', \mu X.\beta(X))$ for some s'. We call such positions *critical*. Consider a tree of all plays starting in $(s, \mu X.\beta(X))$ and respecting the winning strategy for Eve. We can assign to every critical position an ordinal, bounding a number of occurrences of critical positions on the paths starting from it. We call this ordinal the *height* of the position. All critical positions that do not have critical positions in their subtree will have height 1. Then, by induction, we take a critical position p such that all critical positions in its subtree have already a height assigned. Let τ be the least upper bound of these heights. We assign to p the height $\tau + 1$. It is not difficult to see that this procedure will assign a height to every critical position. Figure 5 gives an example of a tree where this procedure needs infinite ordinals.

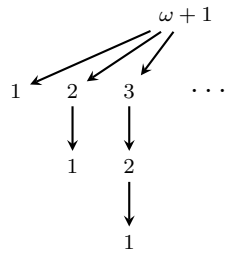

Fig. 5 A transition system where the closure ordinal of $\mu Z.[a]Z$ is infinite

Now, by induction on the ordinals assigned to critical positions we show that if $(s', \mu X.\beta(X))$ has height τ then $\mathcal{M}, s, \mathcal{V} \vDash \mu^{\tau}.\beta(X)$. First take a position $(s', \mu X.\beta(X))$ of height 1. This means that Eve has a winning strategy from this position that never enters another critical position. But then Eve has also a winning strategy from $(s', \beta(X))$ in the game $\mathcal{G}_{\mathcal{V}^1}$ where $\mathcal{V}^1$ is a valuation assigning $\varnothing$ to X. By induction hypothesis $\mathcal{M}, s, \mathcal{V}^1 \vDash \beta(X)$, which is equivalent to $\mathcal{M}, s, \mathcal{V} \vDash \mu^1 X.\beta(X)$. By a similar argument if we take a critical position $(s', \mu X.\beta(X))$ of height $\tau + 1$ then Eve has a strategy winning from $(s', \beta(X))$ in the game $\mathcal{G}_{\mathcal{V}^\tau}$ where $\mathcal{V}^\tau$ is a valuation assigning to X the set of all nodes s'' such that $(s'', \mu X.\beta(X))$ has the height at most τ. By induction hypothesis $\mathcal{M}, s, \mathcal{V}^\tau \vDash \beta(X)$. From the induction on the height we can deduce that $\mathcal{V}^\tau(X) \subseteq [\![\mu^\tau X.\beta(X)]\!]^{\mathcal{M}}_{\mathcal{V}}$. We obtain $\mathcal{M}, s, \mathcal{V} \vDash \mu^{\tau+1} X.\beta(X)$. This completes the induction and the proof.

2.4 Model-checking

The model-checking problem for the mu-calculus is: given a sentence α, a transition system $\mathcal{M}$, and its state s, decide if $\mathcal{M}, s \vDash \alpha$. Formulating the question in such a way, we silently assume that $\mathcal{M}$ is finite and given explicitly.

One approach to model-checking is to use directly the mu-calculus semantics, or its variation through approximations. Since $\mathcal{M}$ is finite, there is a finite number of sets of states of $\mathcal{M}$ and such a computation will terminate.

The semantics via games gives a more efficient way to treat the problem. Observe that the size of the game $\mathcal{G}(\mathcal{M}, \alpha)$ is linear both in the size of $\mathcal{M}$ and in the size of α. So there is a linear time reduction from the model-checking problem to deciding a winner in a parity game. We should remark that the reduction is also linear for formulas in vectorial syntax.

Theorem 7 (Equivalence of games and model-checking [46, 43]). *The model-checking problem is linear-time equivalent to the problem of deciding if Eve has a winning strategy from a given position in a given parity game. The game constructed from a transition system of size m and a formula of size n has size $O(m \cdot n)$, the number of ranks in the game is equal to one more than the alternation depth of the formula. Conversely, from a game one can construct a transition system and a formula. The transition system is of the same size as the game. The formula depends only on the ranks used in the game, its size is linear in the number of ranks, and its alternation depth is not bigger than the number of ranks.*

The increase by 1 of the number of ranks in the game in the above theorem is less disturbing than it looks. Such an increase can appear for formulas that are both in Σ_n^μ and in Π_n^μ classes (cf. page 10). For example, the formula $(\mu X.[a]X) \wedge (\nu Y.\langle a\rangle Y)$ is of alternation depth 1, and the constructed game needs rank 1 for μ, and rank 0 for ν (cf. page 15). This does not really increase the complexity of solving the game as there will be no strongly connected component of the game graph containing both ranks 0 and 1. In general, in every connected component of a game constructed from a transition system and a formula of alternation depth d, the number of ranks in a connected component will be not bigger than d. Games with this property can be solved as quickly as games with d ranks.

The problem of solving parity games is in its turn equivalent to checking emptiness of alternating automata on infinite sequences over a one-letter alphabet. The latter is the same as checking emptiness of nondeterministic tree automata. For definitions of these types of automata we refer the reader to [120, 114, 60] and to the automata chapter of this Handbook [77]. Here we just state the corollary that is a consequence of these equivalences.

Theorem 8 (Equivalence of model-checking and automata emptiness). *Model-checking problem is linear-time equivalent to checking emptiness of alternating parity word automata over a one-letter alphabet. It is also*

equivalent to checking emptiness of nondeterministic parity tree automata. The automata in question have the same number of states as there are nodes in the graph of the game from Theorem 7, and the same parity condition as the game.

Going back to Theorem 7, we briefly discuss the reduction from games to model-checking as the other reduction is already provided by Theorem 6. It turns out that once the ranks are fixed, one can write a formula defining Eve's winning positions in a game with those ranks. Moreover, the size of this formula is linear in the number of ranks. More precisely, let us suppose that the ranks range from 0 to $2d+1$. A game $\mathcal{G} = \langle V, V_E, V_A, T \subseteq V \times V, \Omega : V \to \{0, \ldots, 2d+1\}\rangle$ can be represented as a transition system $\mathcal{M}_\mathcal{G}$ where V is the set of states, and T is a transition relation on some letter, say b. The additional information as to whether a position belongs to V_E, and what is its rank is coded with auxiliary propositions: $p_{Eve}, p_{Adam}, p_0, \ldots, p_{2d+1}$. To write a formula defining the winning position we take variables $Z_0, \ldots, Z_{2d+1}$, one for every possible value of the rank. The formula is:

$$\mu Z_{2d+1}.\nu Z_{2d} \ldots \mu Z_1.\nu Z_0.\ \gamma(Z_0, \ldots, Z_d) \quad \text{where} \tag{1}$$
$$\gamma(Z_0, \ldots, Z_d) \text{ is } \bigwedge_{i=0,\ldots,2d+1} p_i \Rightarrow \big[\big(p_{Eve} \wedge \langle b\rangle Z_i\big) \vee \big(p_{Adam} \wedge [b] Z_i\big)\big]$$

The subformula $\gamma(Z_0, \ldots, Z_d)$ verifies what is the rank of a position using propositions p_i; this determines the fixpoint variable to be used. The formula also makes a choice of using a $\langle b\rangle$ or $[b]$ modality depending on whether the position belongs to Eve or Adam, respectively. The alternation of fixpoints captures the parity condition. It can be verified that the above fixpoint formula defines the set of positions from which Eve has a winning strategy. Indeed the formula is constructed in such a way that the model-checking game $\mathcal{G}(\mathcal{M}_\mathcal{G}, \gamma)$ is essentially the same as $\mathcal{G}$. If the range of Ω function is not as we have assumed, then it is enough to extend the range, write a formula for the extended range, and then remove the unnecessary fixpoint variables.

Hierarchical boolean equations. Games are not the only way to simplify the structure of the model-checking problem. It turns out that the problem is equivalent to checking if a vectorial mu-calculus formula holds in a particularly simple model: the one containing just one state and no transitions. The later problem is known as *hierarchical boolean equations* because the value of a variable is a boolean: it can be either the empty set or the singleton containing the unique state.

Let $\mathcal{M}_0 = \langle \{s\}, \{R_a\}_{a \in Act}, \{P_i\}_{i \in \mathbb{N}}\rangle$ be the transition system with one state s and all transition relations and predicates empty: $R_a = \varnothing$, $P_i = \varnothing$. The meaning of every variable is then either $\varnothing$ or $\{s\}$. All formulas $\langle a\rangle\alpha$ are equivalent to ff, and formulas $[a]\alpha$ are equivalent to tt. Hence essentially we are left with boolean connectives and fixpoints. Relying on the equivalence

between games and model-checking described above, the following fact states the announced reduction.

Fact 9 (Model-checking and hierarchical boolean equations) *For every game $\mathcal{G}$ and its position v there is a vectorial formula $\alpha_{\mathcal{G},v}$, of height equal to the number of positions in $\mathcal{G}$ and size linear in the number of edges in $\mathcal{G}$, such that for the one state transitions system $\mathcal{M}_0$ described above:*
the first coordinate of $[\![\alpha_{\mathcal{G},v}]\!]^{\mathcal{M}_0}$ is equal to $\{s\}$ iff v is winning for Eve in $\mathcal{G}$.

We will briefly explain this reduction. Given a finite game

$$\mathcal{G} = \langle V, V_E, V_A, T \subseteq V \times V, \Omega : V \to \mathbb{N} \rangle$$

we write a vectorial formula of height $n = |V|$. We introduce variables X_i^v where $v \in V$ is a position, and $i \in \Omega(V)$ is one of the possible ranks. Then we define a formula:

$$\alpha^v = \begin{cases} \bigvee \{X^{v'}_{\Omega(v')} : T(v,v')\} & \text{if } v \text{ is a position of Adam,} \\ \bigwedge \{X^{v'}_{\Omega(v')} : T(v,v')\} & \text{if } v \text{ is a position of Eve.} \end{cases}$$

Let us fix some arbitrary order on positions V, assuring that our chosen position is the first one in this order. We get a vectorial formula $\boldsymbol{\alpha} = (\alpha^{v_1}, \ldots, \alpha^{v_n})$. Let $\mathbf{X}_i$ stand for the vector $X_i^{v_1} \ldots X_i^{v_k}$. Then the desired formula is

$$\nu \mathbf{X}_d . \mu \mathbf{X}_{d-1} \ldots . \nu \mathbf{X}_0 . \boldsymbol{\alpha}. \tag{2}$$

Observe that this formula does not have free variables. While the presented translation is superficially quadratic, it should be clear that out of the variables $X_1^v \ldots X_d^v$, only the one with the subscript $\Omega(v)$ is used. So the size of the formula not counting dummy variables is proportional to the number of edges in the game.

Complexity of model-checking. The complexity of model-checking is the great unanswered question about the modal mu-calculus. Effectiveness on finite systems is easy: the obvious algorithm based on the semantics via approximations has complexity $O(n^{d+1})$, where n is the size of the state-space, and d is the alternation depth of the formula. The reduction to parity games quickly gives some new insights. To decide if Eve wins from a given position it is enough to guess a strategy and check that it is winning. By Theorem 5 one can restrict to positional strategies, that are nothing else but subsets of edges of the game graph. It is not difficult see that one can check in a polynomial time if a given positional strategy is winning. The algorithm essentially involves analysis of strongly connected components in the game graph. Consequently, solving the game is in NP, and since everything is closed under negation, also in co-NP. The obvious lower bound is PTIME since alternating reachability is a very simple instance of parity games.

Most of the effort on complexity analysis of the model-checking problem concentrated on the game formulation [48, 46, 43, 84, 125, 72, 122, 18, 73, 103, 20], although it is worth mentioning also the approaches through vectorial syntax and boolean equations [79, 108]. In the discussion below let n stand for the number of vertices in the game and d for the number of ranks. In terms of model-checking problem, n is the product of the sizes of the transition system and the formula, while d is the alternation depth of the formula.

One of the most classical approaches to solve parity games is based on removing positions of the highest rank and recursively analysing the resulting subgame [84, 125, 114]. Since the subgame is analysed twice, this approach gives $O(n^d)$ complexity. A different technique proposed by Jurdziński [72] is based on so called progress measures and gives an $O(n^{\lceil d/2 \rceil})$ algorithm. More recently, Schewe [103] used a combination of the two to obtain $O(n^{\lceil d/3 \rceil})$ algorithm. Better approaches are known if there are many priorities with respect to the number of nodes, more precisely if $d = \Omega(n^{(1/2)+\varepsilon})$. The randomized algorithm of Björklund et al. [18] gives $n^{O(\sqrt{n/\log(n)})}$ complexity. If the out-degree of all vertices is bounded then the same complexity is achieved by a relatively simple, and very clever, modification of McNaughton's algorithm [73] proposed by Jurdziński, Paterson and Zwick. In the case when there is no bound on the degree, the same algorithm is only slightly worse: $n^{O(\sqrt{n})}$. For all of those algorithms superpolynomial, or exponential lower bounds are known [72, 103, 53].

There exists another class of algorithms, based on strategy improvement techniques [66]. The idea is that one puts an order on positional strategies, so that the biggest elements in this order are the optimal strategies. The algorithm is an iteration of improvement steps: in each step some edges of the strategy are changed to improve with respect to the order on strategies. This technique has been used in many contexts. It has been adapted to parity games by Vöge and Jurdziński [122]. Later Schewe [104] has proposed a modification of the strategy improvement policy. Even for this improvement, Friedmann gives examples of games requiring exponential numbers of iterations [52].

It is actually not that surprising that the quest for polynomial time algorithm for model-checking is still on. The problem is closely related to other stubborn questions of a similar type, as for example: solving mean pay-off games, discounted pay-off games, turn based simple stochastic games [71, 36]. Not much more is known about fragments of the logic. The reduction to games gives an easy model-checking algorithm for the alternation depth 2 fragment. This algorithm is quadratic in the size of the formula and the model. No essentially better algorithms are known, but it is worth to mention in this context a recent advance on related problems [32]. Let us note that model-checking for alternation-free mu-calculus (alternation depth 1) can be done in linear time [7, 34, 3].

3 Fundamental properties

In this section we will give an overview of the theory of the mu-calculus. Our intention is to cover a representative selection of results having in mind applications in verification. We will start with some basic results on theory of models: the tree-model property, and bisimulation invariance. Then we will introduce modal automata: an automata model for the mu-calculus. We will discuss some basic features of this model, and in particular disjunctive form for automata. Disjunctive modal automata can be seen as a nondeterministic counterpart of modal automata. Among other things, these automata allow us to treat the satisfiability problem. We present a relatively short proof of EXPTIME-completeness of the problem. On the way we also get a small model theorem. Next, we briefly describe another central result of the theory: strictness of the alternation hierarchy. It is followed by the completeness theorem, and two other properties of more syntactical nature: the interpolation theorem, and the division property.

3.1 Bisimulation invariance and the tree-model property

Bisimulation was introduced in the context of modal logics as an attempt to formulate the notion of similarity between models. Later it turned out that it is perfectly adapted to express the intuition that two systems behave in the same way. Formally, a *bisimulation* between two transition systems

$$\mathcal{M}^1 = \langle S^1, \{R^1_a\}_{a\in Act}, \{P^1_i\}_{i\in\mathbb{N}}\rangle \quad \text{and} \quad \mathcal{M}^2 = \langle S^2, \{R^2_a\}_{a\in Act}, \{P^2_i\}_{i\in\mathbb{N}}\rangle$$

is a relation $\approx$ between S^1 and S^2 such that if $s_1 \approx s_2$ then

- s_1 and s_2 satisfy the same propositions;
- for every action a, and s'_1 such that $R^1_a(s_1, s'_1)$ there is s'_2 with $R^2_a(s_2, s'_2)$ and $s'_1 \approx s'_2$;
- and symmetrically, for every a, and s'_2 such that $R^2_a(s_2, s'_2)$ there is s'_1 with $R^1_a(s_1, s'_1)$ and $s'_1 \approx s'_2$;

We say that a state of $\mathcal{M}^1$ is *bisimilar* to a state of $\mathcal{M}^2$ if there is a bisimulation relating the two states.

The intuition that the mu-calculus is about behaviors finds further support in the following result saying that the logic cannot distinguish between bisimilar states.

Theorem 10 (Invariance under bisimulation). *If a state s_1 of a transition system $\mathcal{M}^1$ is bisimilar to a state s_2 of a transition system $\mathcal{M}^2$ then for every μ-calculus sentence α: $\mathcal{M}^1, s_1 \vDash \alpha$ iff $\mathcal{M}^2, s_2 \vDash \alpha$.*

A consequence of this theorem is the tree-model property. A transition system is a *(directed) tree* if it has a state with a unique path to every other state in the transition system. This special state is called the *root*. A tree can be obtained as an *unfolding* of a transition system from a state s by taking all the paths starting in s as states of the unfolding. In the result, state s of the initial system and the root of the unfolding are bisimilar.

Proposition 1 (Tree models). *Every satisfiable sentence of the mu-calculus is satisfied in a root of some tree transition system.*

Theorem 10 can be deduced from the semantics of the mu-calculus in terms of games. To see this, we define bisimulation relation between games as a bisimulation between transition systems representing games (cf. page 19). A parity game $\mathcal{G} = \langle V, V_E, V_A, T \subseteq V \times V, \Omega : V \to \mathbb{N} \rangle$ can be considered as a transition system with V being a set of states and T the transition relation on some action. The additional information as to whether a position belongs to V_E and the rank of a position are coded with auxiliary propositions: $p_{Eve}, p_{Adam}, p_0, \ldots, p_d$. So bisimulation relation between games will be a bisimulation relation between their representations as transition systems. It is obvious that if a position v_1 in $\mathcal{G}^1$ is bisimilar to a position v_2 in $\mathcal{G}^2$ then Eve has a winning strategy from v_1 iff she has a winning strategy from v_2. The proof of Theorem 10 follows from the fact that if a state s_1 of $\mathcal{M}^1$ is bisimilar to a state s_2 of $\mathcal{M}^2$ then the positions (s_1, α) in $\mathcal{G}(\mathcal{M}^1, \alpha)$ and (s_2, α) in $\mathcal{G}(\mathcal{M}^2, \alpha)$ are bisimilar. This reasoning is a good example of the use of the semantics in terms of games. Let us point out though that Theorem 10 is one of the rare facts in the theory of the mu-calculus that can be proven directly by induction on the syntax of the formula, using approximations in the case of fixpoint formulas (cf. page 5).

3.2 Modal automata

The characterization of the semantics of the mu-calculus in terms of games suggests a kind of operational understanding of the logic. This idea is pushed a bit further here, by introducing an automata model that corresponds to the logic. As we will see the automata model is very close to formulas, but has its technical advantages. First, automata come with a notion of state and this helps in some constructions (cf. Sections 3.3, 3.7). Moreover, modal automata have a very convenient special form called disjunctive modal automata. It is used, for example, to prove the interpolation property (cf. Section 3.6). Modal automata can be also smoothly generalized to give extensions of the mu-calculus with good properties (cf. Section 4.3).

Fix finite sets $\Sigma_A \subseteq Act$ of actions, $\Sigma_P \subseteq Prop$ for propositions, and Q of states. The set of *modal formulas* over these three sets, $\mathcal{F}_m(\Sigma_A, \Sigma_P, Q)$, is the smallest set containing $\Sigma_P \cup Q$ and closed under taking conjunction

$(\alpha \wedge \beta)$, disjunction $(\alpha \vee \beta)$ and two modalities ($\langle b\rangle\alpha$ and $[b]\alpha$), for $b \in \Sigma_A$. Observe that modal formulas are just like mu-calculus formulas but without the fixpoint constructs.

An *modal automaton* is a tuple:

$$\mathcal{A} = \langle Q,\ \Sigma_A,\ \Sigma_P,\ q^0 \in Q,\ \delta : Q \to \mathcal{F}_m(\Sigma_A, \Sigma_P, Q),\ \Omega : Q \to \mathbb{N}\rangle.$$

It has a finite set of states Q, finite action and proposition alphabets Σ_A and Σ_P, one initial state q^0, and the parity acceptance condition given by Ω. The least standard part of the automaton is its transition function: the dependence on the alphabet is hidden in modal formulas.

Example: Let us write an automaton to represent the formula $\mu X.p_1 \vee \langle b\rangle X$. The alphabets of the automaton are $\Sigma_A = \{b\}$, $\Sigma_P = \{p_1\}$. It has one state q, and we let $\delta(q) = p_1 \vee \langle b\rangle q$. Since the formula uses the least fixed point we put $\Omega(q) = 1$.

A modal automaton accepts transition systems with distinguished states. The acceptance is defined by games. Given a transition system $\mathcal{M} = \langle S, \{R\}_{a\in Act}, \{P_i\}_{i\in\mathbb{N}}\rangle$ we consider the *acceptance game* $\mathcal{G}(\mathcal{M}, \mathcal{A})$:

- The set of positions V of the game consist of pairs (s, α) where $s \in S$ is a state of $\mathcal{M}$, and α is a subformula of $\delta(q)$ for some $q \in Q$. A position (s, α) is for Eve if α is of one of the forms: q, $\beta' \vee \beta''$, or $\langle a\rangle\beta'$. Eve also has a position (s, p) if $s \not\models p$, and a position $(s, \neg p)$ if $s \models p$. The remaining positions are for Adam.
- From $(s, \alpha \vee \beta)$ and $(s, \alpha \wedge \beta)$ there are transitions to (s, α) and (s, β). From $(s, \langle a\rangle\alpha)$ and $(s, [a]\alpha)$ there are transitions to (t, α) for all t such that $(s, t) \in R_a$. From (s, q) there is a unique transition leading to $(s, \delta(q))$.
- The rank of a position of the form (s, q) is $\Omega(q)$, all other positions have rank 0.

The automaton *accepts* a transition system $\mathcal{M}$ from the state s iff Eve has a winning strategy in the game $\mathcal{G}(\mathcal{M}, \mathcal{A})$ from the position (s, q^0); recall that q^0 is the initial state of $\mathcal{A}$. We will denote this by $(\mathcal{M}, s) \in L(\mathcal{A})$, so the *language of* $\mathcal{A}$ is the set of transition systems with distinguished states.

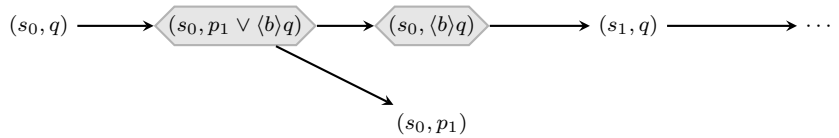

Fig. 6 The acceptance game $\mathcal{G}(\mathcal{M}, \mathcal{A})$ for a modal automaton.

Example: Let us take the automaton $\mathcal{A}$ from the previous example and some structure $\mathcal{M}$. Figure 6 shows how the game $\mathcal{G}(\mathcal{M}, \mathcal{A})$ looks like. Starting in

the position (s_0, q) the unique next position is $(s_0, p_1 \vee \langle b \rangle q)$. If p_1 holds in s_0 then Eve can choose this disjunct and win. Otherwise she chooses $\langle b \rangle q$, and then she chooses a successor s_1 of s_0 by a b-transition. The game reaches the position (s_1, q), and the whole reasoning repeats. It is possible that the game does not end. In this case Eve loses as the rank of q is odd. Observe that for this simple automaton Adam has no choice to make. Hence the automaton accepts iff there is a path of b-transitions leading to a state satisfying p_1.

Example: It may be instructive to see how nondeterministic automata on binary trees [114] can be represented as modal automata. A full labeled binary tree over an alphabet Σ_P is a function $t : \{l, r\}^* \to \Sigma_P$ with the empty word ε being the root and every word $w \in \{l, r\}^*$ being a node with a label $t(w)$, the left successor wl, and the right successor wr. Such a tree can be represented as a transition system $\mathcal{M} = \langle \{l, r\}^*, R_l, R_r, \{p\}_{p \in \Sigma_P} \rangle$. A transition function of a nondeterministic tree automaton has a shape $\delta_{\mathcal{B}} : Q \times \Sigma_P \to \mathcal{P}(Q \times Q)$, that is, to each state and letter it assigns a set of pairs of states: the state to send to the left child, and the state to send to the right child. The corresponding transition function for modal automaton becomes:

$$\delta_{\mathcal{A}}(q) = \bigvee_{p \in \Sigma_P} \left(p \wedge \bigvee_{(q_l, q_r) \in \delta(q,a)} \langle l \rangle q_l \wedge \langle r \rangle q_r \right) .$$

The first element of the conjunct checks what is the label of the node, the second conjunct applies one of the possible transitions.

Equivalence with the mu-calculus. We say that a modal automaton $\mathcal{A}$ is equivalent to a μ-calculus sentence α if for every transition system $\mathcal{M}$ and its state s:

$$(\mathcal{M}, s) \in L(\mathcal{A}) \quad \text{iff} \quad \mathcal{M}, s \vDash \alpha.$$

Thanks to the form of automata, it is easy to show that automata are equivalent to the logic.

Theorem 11 (Modal automata [69]). *For every sentence of the mu-calculus there is an equivalent modal automaton. Conversely, for every modal automaton there is an equivalent sentence of the mu-calculus.*

It is worth looking at the constructions involved in this theorem. For the first statement of the theorem we take a formula α where every variable is bound at most once. This means that for every variable Y bound in α there is precisely one fixpoint subformula of α binding Y, we will denote it by $\sigma Y.\beta_Y$.

We construct an equivalent automaton $\mathcal{A}_\alpha$. Let Σ_A be the set of actions appearing in α, let Σ_P be the set of propositions appearing in α. We take as Q the set containing q^0 and all bound variables appearing in α. The initial state of the automaton is q^0, and the acceptance condition Ω is defined using the notion of alternation depth (cf. Definition 1). The value $\Omega(q_0)$ is irrelevant since the automaton will never come back to q_0; for the other states we have:

$$\Omega(Y) = 2 \cdot \lfloor adepth(Y)/2 \rfloor \qquad \text{if } Y \text{ is bound by } \nu \text{ in } \alpha,$$
$$\Omega(Y) = 2 \cdot \lfloor adepth(Y)/2 \rfloor \quad +1 \qquad \text{if } Y \text{ is bound by } \mu \text{ in } \alpha.$$

It remains to define the transition function of the automaton. For a subformula γ of α, we denote by $\widehat{\gamma}$ the formula obtained by replacing every fixpoint subformula $\sigma Y.\beta_Y$ by the variable Y. So $\widehat{\gamma}$ is a modal formula from $\mathcal{F}_m(\Sigma_A, \Sigma_P, Q)$. With a help of this notation we put:

$$\delta(q_0) = \widehat{\alpha}; \text{ and } \delta(Y) = \widehat{\beta}_Y \text{ for every } Y \in Q.$$

In order to see that the automaton defined in such a way is equivalent to α, it suffices to observe that the evaluation game $\mathcal{G}(\mathcal{M}, \alpha)$ is isomorphic to the acceptance game $\mathcal{G}(\mathcal{M}, \mathcal{A})$; in the former valuation is irrelevant as α is a sentence.

For the second statement of Theorem 11 let us take an automaton $\mathcal{A} = \langle Q, \Sigma_A, \Sigma_P, q^0, \delta : Q \to \mathcal{F}_m(\Sigma_A, \Sigma_P, Q), \Omega : Q \to \mathbb{N} \rangle$. We construct the desired formula in the vectorial syntax. The values of the transition function, $\delta(q)$, are from $\mathcal{F}_m(\Sigma_A, \Sigma_P, Q)$. These are formulas where the elements of Q play the role of variables. We introduce fresh variables X_i^q for every state $q \in Q$, and every possible rank $i \in \Omega(Q)$. Let α^q be the formula $\delta(q)$ with every state q' replaced by $X^{q'}_{\Omega(q')}$. With this notation we set $\boldsymbol{\alpha} = (\alpha^{q_0}, \ldots, \alpha^{q_n})$ where $q_0, \ldots, q_n$ is some enumeration of Q. The formula we are after is $\sigma \mathbf{X}_d \ldots \mu \mathbf{X}_1 . \nu \mathbf{X}_0 . \boldsymbol{\alpha}$; where the least fixed point is used to bind variables with odd index, and the greatest fixed point is used for variables with even index. Using the semantics of formulas in terms of games it is not difficult to show the equivalence of the formula with the automaton.

Closure properties of modal automata. From the above theorem it follows that modal automata are closed under boolean operations because the μ-calculus is. The proof also shows that one can directly use logical laws on transitions of modal automata without changing the accepted language. More precisely, if $\mathcal{A}_1$ and $\mathcal{A}_2$ are two automata over the same alphabet and the same set of states such that for every $q \in Q$, $\delta_1(q)$ is equivalent as a modal formula to $\delta_2(q)$ then $\mathcal{A}_1$ and $\mathcal{A}_2$ accept the same structures. This observation allows us to simplify transitions of automata.

As a side remark let us observe that unlike nondeterministic automata on binary trees, modal automata are not closed under projection. This is not a surprise since we want the automata to be equivalent to the mu-calculus. The automata, and logic, are closed under a weaker form of projection as explained in the interpolation section below.

Disjunctive normal form for automata. Modal automata are essentially alternating automata [90, 77]. This is so because conjunction appearing in modal formulas permits the encoding of universal choice. In some contexts though, nondeterministic automata are much easier to handle than alternating automata. It is well-known that over binary trees every alternating

automaton can be converted to a nondeterministic one. Here we present a similar result for modal automata.

A simple solution to define nondeterministic automata would be to disallow conjunctions in modal formulas. This is not satisfactory though as we would have no way to write properties like $(\langle a\rangle p_1) \wedge (\langle a\rangle p_2)$. There is an interesting modal operator that allows control over the use of conjunction. If Γ is a finite set of formulas and a is an action then

$$(a \to \Gamma) \quad \text{stands for} \quad \Big(\bigwedge\{\langle a\rangle\alpha : \alpha \in \Gamma\}\Big) \wedge [a]\big(\bigvee \Gamma\big).$$

We adopt the convention that the conjunction of the empty set of formulas is equivalent to tt, and its disjunction is equivalent to ff. So for example $(a \to \varnothing)$ says that there are no successors on action a. The new operator can express both existential and universal modalities:

$$\langle a\rangle\alpha \quad \text{is} \quad (a \to \{\alpha, tt\}) \qquad \text{and} \qquad [a]\alpha \quad \text{is} \quad (a \to \{\alpha\}) \vee (a \to \varnothing).$$

This operator was introduced in [69] and independently in [87] in the context of coalgebraic approach to modal logic.

Define a *disjunctive formula* to be a disjunction of formulas of the form $\alpha \wedge \bigwedge_{a\in\Sigma}(a \to \Gamma_a)$ where α is a conjunction of propositions and each Γ_a is a set of states. Let $\mathcal{DF}(\Sigma_A, \Sigma_P, Q)$ denote the set of disjunctive formulas over the three alphabets. A *disjunctive modal automaton* is a modal automaton using disjunctive modal formulas in its transitions.

Theorem 12 (Disjunctive modal automata [69]). *Every modal automaton is equivalent to a disjunctive modal automaton.*

As we have noticed, modal automata may exhibit alternating behavior. Let us see why disjunctive modal automata have only nondeterministic behavior, that is they do not have universal branching. For this we need to look at their behavior on some transition system $\mathcal{M}$ that is a tree. The crucial property is that for every strategy of the automaton in the semantics game induced by $\mathcal{M}$, if two plays respecting the strategy differ at some position then they do not visit the same state of $\mathcal{M}$ after this position. So in the tree of all plays respecting a strategy, no state of $\mathcal{M}$ can appear in two different subtrees. This property corresponds to the fact that a run of a nondeterministic automaton on a binary tree can be presented as a labeling of the tree with states. Observe, incidentally, that the translation of nondeterministic tree automata to modal automata presented in an example on page 25 gives in fact a disjunctive modal automaton. Similarly to nondeterministic automata, the emptiness problem is easier for disjunctive modal automata: it suffices to guess a subgraph of the transition graph of the automaton. Complexity-wise the problem is in NP, while the emptiness problem for modal automata is EXPTIME-complete.

3.3 Satisfiability

Suppose that we want to decide if given two formulas are equivalent. For this we should decide if the two formulas are satisfied in the same set of models; in other words, decide if the logical equivalence of the two formulas holds in every model. Thus formula equivalence is nothing else than the satisfiability problem: deciding if there exists a model and a state where the formula is true. In this subsection we will discuss what is needed to solve the satisfiability problem.

Theorem 13 (Satisfiability [45, 47]). *The satisfiability problem for the μ-calculus is* EXPTIME*-complete.*

Using the correspondence between modal automata and μ-formulas, cf. Theorem 11, instead of studying the satisfiability problem we can study the emptiness problem for modal automata: a formula is satisfiable iff the language of the modal automaton is not empty. Our goal is to reduce the emptiness problem for modal automata to the same problem for alternating automata on binary trees. Although it would be much simpler to use disjunctive automata, we do not follow this path, as we have not discussed how to obtain disjunctive automata. Indeed, a translation to disjunctive automata would contain all the complexity of the satisfiability problem. The argument we present below is an example of an interesting technique of simplifying a problem by enriching transition systems with additional information.

Recall that a modal automaton is a tuple

$$\mathcal{A} = \langle Q, \Sigma_A, \Sigma_P, q^0 \in Q, \delta : Q \to \mathcal{F}_m(\Sigma_A, \Sigma_P, Q), \Omega : Q \to \mathbb{N} \rangle,$$

where $\mathcal{F}_m(\Sigma_A, \Sigma_P, Q)$ is a set of modal formulas over actions in Σ_A, propositions in Σ_P, and with variables in Q. The acceptance is defined in terms of existence of a winning strategy for Eve in the game $\mathcal{G}(\mathcal{M}, \mathcal{A})$; cf. page 24.

Our first step will be to define witnesses for existence of such a winning strategy. A witness will be a deterministic transition system over a bigger alphabet such that all its paths satisfy certain condition (Lemma 1). Then we will encode such a witnesses into a binary tree, and construct an alternating automaton recognizing all encodings of all possible witnesses. This will give a reduction of the satisfiability problem to the emptiness problem for alternating automata on binary trees.

The whole argument crucially depends on the fact that in parity games it is enough to consider positional strategies (Theorem 5). Positions in the acceptance game $\mathcal{G}(\mathcal{M}, \mathcal{A})$ are of the form (s, α) where s is a state of $\mathcal{M}$ and α a subformula of $\delta(q)$ for some $q \in Q$. In particular, the set of formulas appearing in positions of the game is finite. A positional strategy for Eve makes two kinds of choices: (i) in a position of the form $(s, \alpha \vee \beta)$ it chooses (s, α) or (s, β); (ii) in a position of the form $(s, \langle a \rangle \alpha)$ it chooses (s', α) for some state s'. So for a fixed state s the information provided by the strategy is

finite: for every disjunction one disjunct is chosen, for every diamond formula $\langle a\rangle\alpha$ a successor state is chosen. These choices can be encoded in a label of a node, and we will still have only a finite number of labels.

Take a positional strategy σ for Eve in $\mathcal{G}(\mathcal{M},\mathcal{A})$. We introduce new propositions and actions. A proposition $p^{\alpha}_{\alpha\vee\beta}$ will hold in s when $\sigma(s,\alpha\vee\beta)=(s,\alpha)$. A transition on action $b_{\langle a\rangle\alpha}$ from s to s' will mean that $\sigma(s,\langle a\rangle\alpha)=(s',\alpha)$. Let $\widehat{\Sigma}_P$ and $\widehat{\Sigma}_A$ be the alphabets of these new propositions and actions. Let $\sigma(\mathcal{M})$ stand for the transition system obtained from $\mathcal{M}$ by adding the new propositions and transitions in the way we have described.

We claim that looking at $\sigma(\mathcal{M})$ we can decide if σ is winning in $\mathcal{G}(\mathcal{M},\mathcal{A})$. To make it precise we define a notion of a *trace*. It is a sequence of pairs (s,α) consisting of a state of $\sigma(\mathcal{M})$ and a formula, such that:

- (s,q) is followed by $(s,\delta(q))$;
- $(s,\alpha\wedge\beta)$ is followed by (s,α) or (s,β);
- $(s,\alpha\vee\beta)$ is followed by (s,α) if $p^{\alpha}_{\alpha\vee\beta}$ holds in s, and by (s,β) otherwise;
- $(s,\langle a\rangle\alpha)$ is followed by (s',α) if there is transition from s to s' on action $b_{\langle a\rangle\alpha}$;
- $(s,[a]\beta)$ is followed by (s',β) if for some α there is transition from s to s' on action $b_{\langle a\rangle\alpha}$.

Examining this definition one can observe that a trace in $\sigma(\mathcal{M})$ is just a play in $\mathcal{G}(\mathcal{M},\mathcal{A})$ respecting the strategy σ. So we can say that a trace is *winning* if it is winning when considered as a play in $\mathcal{G}(\mathcal{M},\mathcal{A})$. Finally, we say that $\sigma(\mathcal{M})$ is *trace-accepting from* (s^0,q^0) if all the traces starting in (s^0,q^0) are winning.

The first observation is that without loss of generality we can assume that $\sigma(\mathcal{M})$ is *deterministic*, in a sense that from every state, there is at most one outgoing transition on each action. Indeed, if $\sigma(\mathcal{M})$ is trace accepting and it has a state with two outgoing transitions on action $b_{\langle a\rangle\alpha}$ then we can just remove one of them. This operation cannot add new traces, so the resulting structure is trace-accepting: the structure is even of the form $\sigma'(\mathcal{M})$ for some strategy σ'.

Lemma 1. *For our fixed modal automaton $\mathcal{A}$ and alphabet $\widehat{\Sigma}$: there is a transition system accepted by $\mathcal{A}$ iff there is a deterministic transition system over $\widehat{\Sigma}$ that is trace-accepting.*

We have defined $\sigma(\mathcal{M})$ and a notion of a trace in such a way that if σ is a positional winning strategy in $\mathcal{G}(\mathcal{M},\mathcal{A})$ then $\sigma(\mathcal{M})$ is trace accepting. As noted above $\sigma(\mathcal{M})$ can be made deterministic. This shows left to right implication of the lemma.

For the converse implication, let us take a deterministic transition system $\mathcal{N}$ over the alphabet $\widehat{\Sigma}$. We define the structure $\mathcal{M}'$ by keeping the states of $\mathcal{N}$ and putting a transition on a from s to s' if there is a transition form s to s' on $b_{\langle a\rangle\beta}$ for some formula β. The positional strategy σ' in the game $\mathcal{G}(\mathcal{M}',\mathcal{A})$ can be read from $\mathcal{N}$ as follows: $\sigma'(s,\alpha\vee\beta)=\alpha$ if s satisfies $p^{\alpha}_{\alpha\vee\beta}$ in

$\mathcal{N}$, and $\sigma'(s, \langle a \rangle \alpha) = (s', \alpha)$ if there is a transition from s to s' on $\langle a \rangle \alpha$. It can be verified that $\sigma(\mathcal{M}')$ is isomorphic to $\mathcal{N}$. Moreover every play in $\mathcal{G}(\mathcal{M}', \mathcal{A})$ respecting σ' and starting from a position (s, q) is a trace in $\mathcal{N}$ starting in (s, q). Hence if $\mathcal{N}$ is trace-accepting from (s^0, q^0) then σ' is winning from (s^0, q^0). Since q^0 is the initial state of $\mathcal{A}$, this means that the pair $(\mathcal{M}', s^0)$ is accepted by $\mathcal{A}$.

The above lemma permits us to reduce the satisfiability problem to the problem of finding a deterministic structure $\mathcal{N}$ over the alphabet $\widehat{\Sigma}$ that is trace-accepting. Observe that if $\mathcal{M}$ is a tree then $\sigma(\mathcal{M})$ is also a tree. Hence, by the tree model property, Corollary 1, if there is such a structure $\mathcal{N}$ then there is one that is a deterministic tree. Since the set of labels of transitions is finite, $\mathcal{N}$ is a tree of bounded degree.

We have reduced the satisfiability problem to the problem of finding a tree of bounded degree satisfying trace-acceptance condition. Using a straightforward encoding of bounded degree trees into binary trees we can reduce the problem to a question about full binary trees. So finally we need to construct an automaton recognizing binary trees that are encodings of bounded degree trees satisfying trace acceptance condition. Such an automaton $\mathcal{B}_3$ can be constructed in three steps: (i) taking a simple nondeterministic parity automaton $\mathcal{B}_1$ recognizing paths having a trace that is not winning; (ii) dualizing $\mathcal{B}_1$ to obtain an alternating parity automaton $\mathcal{B}_2$ recognizing paths on which all traces are winning; and (iii) constructing an alternating parity tree automaton $\mathcal{B}_3$ running $\mathcal{B}_2$ on every path of a tree.

Theorem 14 (Satifiability via automata emptiness [46]). *For a given mu-calculus formula of size n and alternation depth d, one can construct in linear time an alternating automaton over trees such that the formula is satisfiable iff there is a tree accepted by the automaton. The automaton has $O(n)$ states and $d + 1$ ranks.*

This theorem gives an algorithmic solution to the satisfiability problem. As the emptiness check for such automata can be done in EXPTIME, this gives an upper bound on the complexity of the satisfiability problem as announced in Theorem 13. The matching lower bound can be obtained, for example, by encoding of the universality problem for root-to-leaves automata over finite trees [107].

The above construction proves also the small model theorem. A regular tree can be presented as a graph with a distinguished root vertex: the tree is obtained by *unfolding* such a graph, that is taking all the paths starting in the root. If an alternating automaton accepts some tree then it accepts a regular tree that is the unfolding of a transition system of size exponential in the size of the automaton.

Theorem 15 (Small-model property [113]). *A satisfiable formula of the mu-calculus is satisfied in a transition system of size exponential in the size of the formula.*

3.4 Alternation hierarchy

We have seen that the alternation depth (cf. Definition 1) of a formula appears to give a strong measure of its complexity, both psychological and computational: formulas rapidly become incomprehensible above alternation depth 2, and all algorithms known so far depend exponentially or super-polynomially on the alternation depth (cf. Section 2.4). Recall the definition of sets Σ_n^μ and Π_n^μ from page 10. Roughly, Σ_n^μ is the set of formulas of alternation depth n where all variables of alternation depth n are μ-variables. Similarly for Π_n^μ but for ν-variables.

For a number of years, it was open whether 'the alternation hierarchy is strict' in terms of expressive power, that is, whether for every n there is a Σ_n^μ formula that is not equivalent to any Π_n^μ formula – and consequently, whether alternation depth $n+1$ is strictly more expressive than alternation depth n. The first proof by Bradfield [28] used the standard technique of diagonalization, via a transfer to arithmetic, relying on the small model property for the transfer back. Subsequently, Arnold [6] gave a version of the proof in which diagonalization was effected with a topological argument – as this is probably the simplest proof, we will sketch it here.

Theorem 16 (Strictness of the alternation hierarchy [28]). *For every $n > 0$, there is a formula in Σ_n^μ which is not equivalent to any formula of Π_n^μ.*

The proof relies on the semantics of formulas in terms of games (Section 2.3). Recall that by Theorem 6 for a transition $\mathcal{M}$ and a formula β we can construct a game $\mathcal{G}(\mathcal{M}, \beta)$ such that $\mathcal{M}, s \vDash \beta$ if and only if Eve has a winning strategy from the node (s, β) in $\mathcal{G}(\mathcal{M}, \beta)$. If $\beta \in \Sigma_n^\mu$ then $\mathcal{G}(\mathcal{M}, \beta)$ is a parity game over the ranks $\{1, \ldots, n\}$ or $\{0, \ldots, n-1\}$; depending if n is odd or even, respectively. Let us suppose that it is $\{1, \ldots, n\}$, the argument for the other case is very similar. As we have discussed on page 19, the game $\mathcal{G}(\mathcal{M}, \beta)$ can be represented as a transition system. A small but crucial observation is that if we start with a sufficiently large alphabet then we can actually suppose that this new transition system is over the same alphabet as $\mathcal{M}$. Hence a formula β determines a function F_β on transition systems over some fixed alphabet: for a transition system $\mathcal{M}$, the system $F_\beta(\mathcal{M})$ is the transition system representing the game $\mathcal{G}(\mathcal{M}, \beta)$.

We have supposed that games we consider are over ranks $\{1, \ldots, n\}$. Now recall the formula from (1) on page 19 defining the set of winning positions for Eve in a parity game over priorities $\{0, \ldots, n\}$. Let us call this formula α_n. It means that for every position (s, γ) in game $\mathcal{G}(\mathcal{M}, \beta)$: formula α_n holds in (s, γ) iff (s, γ) is winning for Eve. Combining this with the previous paragraph we get that for every transition system $\mathcal{M}$ and formula $\beta \in \Sigma_d^\mu$: $\mathcal{M}, s \vDash \beta \quad \text{iff} \quad F(\mathcal{M}), (s, \beta) \vDash \alpha_n$. For our argument we will need a slightly more refined construction that works on tree transition systems: transition systems whose underlying graph is a tree. We fix a sufficiently big alphabet,

a variation of formula α_n adapted to this alphabet, and construct a function F'_β such that for every tree transition system $\mathcal{N}$:

$$\mathcal{N}, root \vDash \beta \quad \text{iff} \quad F'_{\mathcal{B}}(\mathcal{N}), root \vDash \alpha'_n \ ;$$

here *root* stands for the root node of a tree transition system. We will omit exact definition of F'_β. Formula α_n belongs to Σ^μ_n, and the same will be true for formula α'_n. We will show that it cannot belong to Π^μ_n.

Trees can be equipped with the usual metric: trees are 2^{-n} apart if they first differ at a node of depth n. The set of infinite trees with this metric is a complete metric space. So the mapping F'_β described above is a mapping on a complete metric space. It turns out that this mapping is contracting. The Banach Fixed Point Theorem says that every contracting mapping on a complete metric space has a fixpoint. In our case it means that for every formula β there is a tree transition system $\mathcal{N}_\beta$ such that $\mathcal{N}_\beta = F'_\beta(\mathcal{N}_\beta)$.

We want to show that $\alpha_n \notin \Pi^\mu_n$. Suppose conversely. Then there is a formula $\gamma \in \Sigma^\mu_n$ equivalent to $\neg\alpha_n$. We consider the mapping F'_γ and its fixpoint $\mathcal{N}_\gamma$. We get $\mathcal{N}_\gamma, root \vDash \gamma$ iff $F'_\gamma(\mathcal{N}_\gamma), root \vDash \alpha_n$. But $\mathcal{N}_\gamma$ is a fixpoint of F'_γ so $F'_\gamma(\mathcal{N}_\gamma) = \mathcal{N}_\gamma$; a contradiction.

This proof shows concrete examples of formulas at each level of the hierarchy: the formulas α_n expressing existence of a winning strategy. Nevertheless the hierarchy is far from being well understood. We do not know for example if the semantic alternation depth, the smallest alternation depth of an equivalent formula, is a decidable property. This is an area of active research, that mostly is formulated in terms of automata theory rather than the mu-calculus.

3.5 Proof system

In order to show that a formula is satisfiable it is enough to exhibit a model for it. Theorem 15 tells us that every satisfiable formula has a finite model. So there is always a finite witness of satisfiability. The question arises, what about unsatisfiability? Since satisfiability is a decidable property (Theorem 13) the run of the algorithm is such a witness. A proof system gives a much more informative witness, moreover it gives reasoning principles that can be used to simplify formulas. To look at the question more positively, one prefers to talk about validity instead of unsatisfiability: a formula is *valid* if its negation is not satisfiable.

Validity of mu-calculus formulas can be axiomatized. It means that for a valid formula one can provide a finite witness, a proof, of its validity. To describe the proof system we will use inequalities $\alpha \leq \beta$. We will say that such an inequality is *valid* if the implication $\alpha \Rightarrow \beta$ is valid. In other words, whenever under some valuation formula α is true in some state of a model

then β is true too in the same state with the same valuation. For example, a formula α is valid iff $tt \leq \alpha$ is valid; and a formula β is satisfiable iff $\beta \leq ff$ is not valid. We present a finitary proof system allowing us to deduce all valid inequalities. As examples of useful inequalities consider:

$$\mu X.\alpha \leq \nu X.\alpha \qquad \mu X.\nu Y.\alpha \leq \nu Y.\mu X.\alpha$$

The proof system proposed by Kozen [76] consists of axioms and rules for modal logic together with an axiom and a rule determining the semantics of the least fixpoint:

$$\alpha(\mu X.\alpha) \leq \mu X.\alpha$$
$$\frac{\alpha[\beta/X] \leq \beta}{\mu X.\alpha(X) \leq \beta}$$

The last rule expresses rather directly the semantic clause defining the least fixpoint (cf. page 4). Additionally to these two, there is a dual axiom and a rule for the greatest fixpoint operator. This system is finitary as it contains only a finite number of axioms and rule schemes. The system is sound and complete in a sense that it proves exactly the valid inequalities.

Theorem 17 (Completeness [123]). *For every formula α: there is a proof of $tt \leq \alpha$ in the system if and only if the formula is valid.*

3.6 Interpolation property

Craig interpolation is a desirable property of a logic. It testifies some kind of adequacy between the syntax of the logic and its semantics. On a more practical side, it allows us to simplify formulas by just looking at their vocabulary [83]. Interpolation properties are scarce. Many program logics, such as LTL, CTL, CTL* for example, do not have the interpolation property [80]. Propositional logic, and modal logic do have interpolation property. So does the mu-calculus.

In order to simplify the presentation, let us restrict here to the mu-calculus where the only propositions are tt and ff. In this case, the Craig interpolation property says that given two formulas α and β over sets of actions Σ_1 and Σ_2 respectively, if $\alpha \Rightarrow \beta$ is valid then there is a formula γ over the set of common actions $\Sigma_1 \cap \Sigma_2$ such that $\alpha \Rightarrow \gamma \Rightarrow \beta$. Actually even a stronger version of interpolation holds for the mu-calculus. In this version γ does not depend on β but only on its alphabet.

Theorem 18 (Interpolation [37]). *Given a formula α over the set of actions Σ_1 and another set of actions Σ_2, there is a formula γ over $\Sigma_1 \cap \Sigma_2$*

such that $\alpha \Rightarrow \gamma$ is valid and for every formula β over the set of actions Σ_2: if $\alpha \Rightarrow \beta$ is valid then $\gamma \Rightarrow \beta$ is valid.

The construction of γ as required in the theorem is very straightforward once we have a disjunctive automaton for α (Theorem 12). For every action a not appearing in $\Sigma_1 \cap \Sigma_2$ it is simply enough to replace every formula $(a \to \Gamma)$ by tt if the formula is satisfiable and by ff otherwise.

3.7 Division property

We want to present one more interesting closure property of the mu-calculus. The motivation comes from modular verification and synthesis. As in the previous subsection, we will consider mu-calculus formulas with only two propositional letters: tt and ff. This restriction simplifies the notion of a product of transition systems. In a product $\mathcal{M} \times \mathcal{N}$ the states are pairs (s_m, s_n) of states of the two systems. We have a transition from (s_m, s_n) to (s'_m, s'_n) on a letter a iff there is one from s_m to s'_m and one from s_n to s'_n. As we do not have propositions we do not need to say which propositions hold in (s_m, s_n).

Imagine that we fix a transition system $\mathcal{M}$ together with a formula α and face the task: given a transition system $\mathcal{N}$ verify if $\mathcal{M} \times \mathcal{N} \vDash \alpha$. That is to verify if the product of a fixed system with a system given on the input satisfies the fixed formula. A straightforward way to solve this problem is to construct the product $\mathcal{M} \times \mathcal{N}$ and then apply a model-checking algorithm. It is possible though to do some pre-processing and construct a formula $\alpha/_{\mathcal{M}}$ as stated in the following theorem.

Theorem 19 (Division operation [4]). *For every transition system $\mathcal{M}$ and every mu-calculus formula α, there is a mu-calculus formula $\alpha/_{\mathcal{M}}$ such that for every transition system $\mathcal{N}$:*

$$\mathcal{N} \vDash \alpha/_{\mathcal{M}} \quad \textit{iff} \quad \mathcal{M} \times \mathcal{N} \vDash \alpha \ .$$

The construction of $\alpha/_{\mathcal{M}}$ is particularly short using the formalism of modal automata (cf. Section 3.2). So below we will talk about $\mathcal{A}/_{\mathcal{M}}$ instead of $\alpha/_{\mathcal{M}}$. Consider a modal automaton $\mathcal{A} = \langle Q, \Sigma, q_0, \delta : Q \to \mathcal{F}(\Sigma, Q), \Omega \rangle$ and a transition system $\mathcal{M} = \langle S, \{R_a\}_{a \in \Sigma} \rangle$, the two over the same set of actions. In the two we just have propositions tt and ff; this justifies writing $\mathcal{F}(\Sigma, Q)$ instead of $\mathcal{F}(\Sigma, \{tt, ff\}, Q)$. Our goal is to construct an automaton $\mathcal{A}/_{\mathcal{M}}$.

We first define a division α/s for α a formula from $\mathcal{F}(\Sigma, Q)$, and s a state of $\mathcal{M}$. The result is a formula from $\mathcal{F}(\Sigma, Q \times S)$:

$$q/s = (q, s)$$
$$(\alpha \vee \beta)/s = \alpha/s \vee \beta/s \qquad (\alpha \wedge \beta)/s = \alpha/s \wedge \beta/s$$
$$(\langle a\rangle\alpha)/s = \langle a\rangle \bigvee\{\alpha/s' : s \xrightarrow{a} s'\} \qquad ([a]\alpha)/s = [a] \bigwedge\{\alpha/s' : s \xrightarrow{a} s'\}$$

The set of states of $\mathcal{A}/_{\mathcal{M}}$ will be $Q_{/} = Q \times S$. The rank function will be inherited from $\mathcal{A}$: $\Omega_{/}(q, s) = \Omega(q)$. Finally, the transition function will be defined using the above operation: $\delta_{/}(q, s) = \delta(q)/s$. Recall that $\delta(q) \in \mathcal{F}(\mathcal{S}, Q)$, so $\delta(q)/s \in \mathcal{F}(\mathcal{S}, Q \times S)$ as required. Once again, the semantics in terms of games gives the tools to prove correctness of the construction. One can show that Eve can win in $\mathcal{G}(\mathcal{N}, \mathcal{A}/\mathcal{M})$ iff she can win in $\mathcal{G}(\mathcal{M} \times \mathcal{N}, \mathcal{A})$.

The division operation lets us solve some kinds of synthesis problems. Suppose that we are given a finite transition system $\mathcal{M}$ modeling behaviors of a device. We want to restrict these behaviors so that the result satisfies a specification given by a formula α. The restriction is modeled by taking a product of $\mathcal{M}$ with another transition system $\mathcal{C}$. This is a restriction in a sense that every path in $\mathcal{M} \times \mathcal{C}$ is a path in $\mathcal{M}$. Transition system $\mathcal{C}$ is considered to be a controller for $\mathcal{M}$. One may also want to put some constraints on $\mathcal{C}$. For example one can single out a set of uncontrollable actions and demand that these actions cannot be restricted by $\mathcal{C}$. This constraint translates to the requirement that from every state of $\mathcal{C}$ there is a transition on every unobservable action. Hence, this condition can be written as a mu-calculus formula β_{unc}. The synthesis problem then becomes: given $\mathcal{M}$ and α find $\mathcal{C}$ such that

$$\mathcal{M} \times \mathcal{C} \vDash \alpha \quad \text{and} \quad \mathcal{C} \vDash \beta_{unc}.$$

Thanks to Theorem 19 the latter is equivalent to

$$\mathcal{C} \vDash (\alpha/_{\mathcal{M}}) \wedge \beta_{unc}.$$

Thus finding a controller $\mathcal{C}$ is reduced to the satisfiability problem for the mu-calculus (Theorem 13). Let us remark that the Church synthesis problem [33, 117] for the mu-calculus formulas is an instance of the above for a particular choice of $\mathcal{M}$.

Not all important constraints though can be expressed in the mu-calculus. For example one can ask that some actions of the system are invisible to a controller. This translates to the requirement that in $\mathcal{C}$ transitions on these actions should be self-loops. This requirement is not invariant under bisimulation, and in consequence is not expressible in the mu-calculus (Theorem 10). Fortunately, it turns out that similar constructions to the above work for the mu-calculus with the self-loop predicate [9], and that the extended logic is still decidable in EXPTIME.

4 Relations with other logics

In this section we look at the mu-calculus in a wider context. Monadic second-order logic (MSOL) is a reference for all temporal and program logics because it is a classical formalism capturing recognizability on words and trees [114]. We will explain why the mu-calculus is closely related to MSOL, and why it is in some sense is the strongest behavioral logic included in MSOL. It is no surprise then that other well-known temporal and program logics can be translated to the mu-calculus. We briefly present the translations of some of them, and discuss their properties. In order to give a broader picture, we briefly describe two interesting extensions of the mu-calculus. One has a semantical flavor: we add some structure on successors of a node in a transition system. The other is more syntactic: we add a fragment of first-order logic into the mu-calculus.

4.1 MSOL, binary trees and bisimulation invariance

A transition system $\mathcal{M} = \langle S, \{R_a\}_{a\in Act}, \{P_i\}_{i\in\mathbb{N}}\rangle$ can be considered as a model of first-order logic, over the signature consisting of binary relations R_a and unary relations P_i. This logic is not sufficiently expressive for verification as it cannot express such a fundamental property as reachabilty: there is a path from x to y. For this and many other reasons [114] it is interesting to consider *monadic second-order logic*, MSOL. This is an extension of the first-order logic with set variables, denoted $X, Y, \ldots$, the membership predicate $y \in X$, and quantification using set variables $\exists X.\varphi$. For example the formula

$$\forall X.\ \big[(y \in X) \wedge \forall z, z'.\ (z \in X \wedge R_b(z, z')) \Rightarrow z' \in X\big] \Rightarrow (y' \in X),$$

expresses that y' is reachable from y by a sequence of b actions. The formula literally says that for every set X if y is in X, and X is closed under taking successors with respect to b actions then y' is in X.

We write $\mathcal{M}, V \vDash \varphi$ to say that an MSOL formula φ holds in the transition system $\mathcal{M}$ under valuation V. Observe that V assigns states of $\mathcal{M}$ to first-order variables in φ and sets of states of $\mathcal{M}$ to set variables in φ. If s is a state of $\mathcal{M}$, and $\varphi(x)$ is a formula with unique free first-order variable x then we simply write $\mathcal{M}, V \vDash \varphi(s)$ for $\mathcal{M}, V[x \mapsto s] \vDash \varphi(x)$, i.e., to say that $\varphi(x)$ is true under a valuation that maps x to s.

Since the satisfiability problem for first-order logic over transition systems is undecidable, so is the one for MSOL. The situation is much more interesting for tree transition systems. Rabin's theorem [99, 114] says that MSOL over tree transition systems is decidable. So it is natural to try to compare its expressive power with that of the mu-calculus over trees. To this end we need to find a common ground for the two logics. A small problem we need

to overcome is that MSOL talks about the truth in a tree, while the mu-calculus about the truth in a node of the tree. Then also we need to take care of first-order variables that are allowed in MSOL, but not in the mu-calculus. We will discuss these points in more detail.

It is quite straightforward to translate the mu-calculus to MSOL. The translation goes by induction on the structure of the formula. For a formula α of the mu-calculus we construct a formula $\varphi(x)$ of MSOL with the same second order variables and one free first-order variable x. The translation has the property that for every transition system $\mathcal{M}$, its state s, and valuation of second-order variables V we have:

$$\mathcal{M}, s, V \vDash \alpha \qquad \text{iff} \qquad \mathcal{M}, V \vDash \varphi(s). \tag{3}$$

For example, we translate a variable Y of the mu-calculus to a formula $x \in Y$. Similarly, a proposition p is translated to $P(x)$. We translate boolean connectives to the same boolean connectives. For modalities and fixpoints we express semantic clauses from page 4 in MSOL.

The translation in the other direction is not that evident. First of all, we restrict to formulas with only one free first-order variable x. So below when we talk about MSOL we consider only such formulas. We say that such a formula is *equivalent* to a mu-calculus formula α if the equivalence (3) holds. Observe that MSOL can express properties that are not bisimulation invariant, like for example "a node has three successors" or "there is a cycle". Since the mu-calculus can express only bisimulation invariant properties (Theorem 10), it cannot be equivalent to MSOL over transition systems. This observation justifies the restriction to deterministic tree transition systems where every successor has a unique name, and there are no cycles.

Theorem 20 (Equivalence with MSOL [91]). *Over deterministic tree transition systems MSOL is equivalent to the mu-calculus.*

An interesting variation of MSOL, called *weak-MSOL*, is obtained by restricting the set variables to range over finite sets only. Another interesting class is that of Σ_1 MSOL formulas, which have a prefix of (unrestricted) existential quantifiers followed by a formula without second-order quantifiers. Rabin has shown [100] that an MSOL formula is equivalent to a weak-MSOL formula iff both a formula and its negation are equivalent to a Σ_1 MSOL formula. He has also shown that Σ_1 MSOL formulas are equivalent to tree automata with Büchi acceptance conditions. It turns out that weak-MSOL is equivalent to the alternation-free fragment of the mu-calculus (which we recall is the fragment consisting of formulas of alternation depth 1, cf. page 9).

Theorem 21 (Equivalence with weak-MSOL [91]). *Over deterministic tree transition systems: weak-MSOL is equivalent to alternation-free mu-calculus. The Σ_1 fragment of MSOL is equivalent to Π_2^μ fragment of the mu-calculus.*

If we drop the determinacy restriction then there is a simple extension of the mu-calculus that captures MSOL on tree transition systems. It is enough to introduce counting modalities $\langle b\rangle_{=n}\alpha$ for every action b and natural number n. The meaning of this modality is that there are exactly n distinct b-successors of a node satisfying α. This observation is an instance of a more general framework described in Section 4.3.1.

Let us come back to the case of transition systems with no restrictions. We have noted above that if an MSOL formula $\varphi(x)$ is equivalent to the mu-calculus formula then it is *bisimulation invariant*: if a formula holds in a state then it holds in every state bisimilar to it. It turns out that the converse implication holds.

Theorem 22 (Expressive completeness [70]). *The mu-calculus is expressively equivalent to the bisimulation invariant fragment of MSOL: if an MSOL formula $\varphi(x)$ is bisimulation invariant then it is equivalent to a mu-calculus formula.*

Recall that the satisfiability problem for MSOL over transition systems is not decidable [42]. In consequence, it is not decidable if a given MSOL formula is bisimulation invariant. So it is not decidable if an MSOL formula can be written in the mu-calculus.

4.2 Embedding of program logics

The mu-calculus is one of the numerous program logics designed to express properties of transition systems. Theorem 22 implies that the mu-calculus is as expressive as any logic that is at the same time bisimulation invariant, and not more expressive than MSOL. This covers most program logics as for example: propositional dynamic logic (PDL), computational tree logic (CTL) and its extension CTL* [51, 44]. It is anyway worthwhile to see explicit translations of these logics into the mu-calculus and discuss their relative expressive powers.

Translating PDL or CTL to the mu-calculus is easy. For sake of example let us look at CTL. The formulas of this logic are built from *tt* and *ff* using boolean connectives, the $\langle a\rangle$ modality, and two operators:

$$\mathsf{E}(\alpha\mathsf{U}\beta) \qquad \mathsf{E}\neg(\alpha\mathsf{U}\beta)$$

As for the mu-calculus, the meaning of a CTL formula is a set of states. The only construct with non-obvious semantics is the until operator:

$\mathcal{M}, s \vDash \mathsf{E}(\alpha\mathsf{U}\beta)$ iff there is a path $s_0, s_1, \ldots, s_k$ such that $s_0 = s$, $s_k \vDash \beta$, and $s_i \vDash \alpha$ for $i = 0, \ldots, k-1$.

For the translation to the mu-calculus, denoted $[\alpha]^\natural$, we just need to take care of the two new operators. We show the translation only for the first one:

$$[\mathsf{E}(\alpha \mathsf{U} \beta)]^\natural \quad \text{is} \quad \mu X.[\beta]^\natural \vee ([\alpha]^\natural \wedge \bigvee_{a \in \Sigma} \langle a \rangle X).$$

The translation produces an alternation-free formula and is linear in size. The translation for PDL is equally easy, but to get linear size formulas we need to use vectorial syntax of the mu-calculus (cf. page 8).

Translating linear time logics, like LTL, is much more complicated. First one needs to make LTL express properties of transition systems. This can be done easily by saying that a formula is true in a state if it is true on all maximal paths from that state. For example, a formula is true in a structure with one state and self-loops on every action iff it is true on every infinite path. Since validity of LTL formulas is PSPACE-complete, this indicates that one can expect an exponential blowup when translating from LTL to the mu-calculus. It is easy to write a mu-calculus formula that is equivalent to LTL over linear models, i.e., transition systems consisting of only one path. For the general case, the problem is that mu-calculus does not have an explicit path quantifier, and there is no way to say easily "every path satisfies some formula". The solution for translating LTL is more complicated. One translates an LTL formula to a Büchi automaton accepting the sequences not satisfying the formula (cf. [55] or [77] in this Handbook). Then this automaton is translated into a mu-calculus formula over linear models. It can be shown that such a formula can be directly used to express the property that there is a path in a transition system not satisfying the initial LTL formula. The negation of this formula is the desired translation. Since CTL* is a common extension of both CTL and LTL, its translation to the mu-calculus combines the two translations described above [39, 17]. One can show that CTL* is contained in Σ_3^μ level of the alternation hierarchy. In the translation described above one can use nondeterministic Büchi automata instead of parity automata. This would give a Π_2^μ formula for every path quantifier. Due to negation and other constructs of CTL*, the complete translation gives a combination of Π_2^μ and Σ_2^μ formulas.

Fact 23 (Embedding of program logics) *Every formula of CTL, PDL, or CTL* can be translated into an equivalent mu-calculus formula.*

Concerning expressiveness, CTL is quite a weak logic. It cannot express for example that there is a path with infinitely many b events on it. This follows from the strictness of the mu-calculus hierarchy (Theorem 16) as CTL can be translated into alternation-free mu-calculus. Even though CTL* can express the "infinitely many" property, it is still expressively weaker than the mu-calculus. For example, it cannot express properties describing infinite interaction of the type "there is a way of repeatedly choosing an output for a given input so that the resulting infinite sequence of inputs and outputs satisfies some given LTL property". In particular, CTL* cannot express all game formulas (1) on page 19, as these formulas can express properties arbitrarily high in the alternation hierarchy (cf. page 31).

The last remark provided an inspiration for the introduction of alternating-time temporal logic [2]. It has been defined over a large class of so called game models, so it is not immediate to compare it to the mu-calculus. Yet, in the cases when alternating-time temporal logic behaves well, the logic can be translated to the mu-calculus. Another interesting logic is game logic [94]. This logic can be translated to the mu-calculus with only two variables, yet it can express properties arbitrarily high in the alternation hierarchy [13].

4.3 Beyond mu-calculus

Because of its simple formulation, its expressive power, and its algorithmic properties, the mu-calculus has proved to be a very valuable logic for verification. As every system, the logic has its limitations, and some of them we have already mentioned in this chapter. The mu-calculus does not allow any form of equality: we cannot say that a transition is a self loop, or that transitions on a and b go to the same state. It permits no form of counting: we cannot say that there is only one a transition from a state. It does not have quantification: we cannot say that a formula holds in every state. It does not allow derived transition relations: we cannot talk about the reverse of a transition relation. Obviously, such a wish list has no end, but it provides good motivation in a search for extensions of the mu-calculus. Of course it is not very difficult to define very expressive logics. What is important is to find extensions without losing good properties of the logic, or at least not all of them.

In this subsection we will present two extensions of the mu-calculus of somehow different natures. The first starts from model-theoretic ideas, the second is motivated by the syntax. The two extensions are quite different in what they accomplish, but both proved to be very useful.

4.3.1 Iterated structures

Till now we have considered logics over transition systems. Here we would like to consider what happens when we put some structure on the set of successors of a state of a transition system. For example, what would happen if we added an order between successors. Instead of considering particular cases we will introduce a general method of constructing transition systems with additional structure. Then we will discuss how to handle this added structure in extensions of the mu-calculus. For this we will define automata that are a generalization of modal automata. We will present their closure properties as well as the relation to MSOL.

Let $\mathcal{N} = \langle D, r_1, \ldots \rangle$ be a structure over a relational signature; this means that functional symbols are not allowed. For example: $\langle \mathbb{N}, \leq \rangle$ is a structure

over a signature containing one relation, the standard ordering; the structure $\langle \mathbb{N}, + \rangle$ is allowed when $+$ is considered as ternary addition relation, and not as a binary function.

An *iterated structure* is a structure $\mathcal{N}^* = \langle D^*, \mathit{child}, \mathit{clone}, r_1^*, \ldots \rangle$ where D^* is the set of all finite sequences over D, and the relations are defined by:

$$\begin{aligned} \mathit{child} &= \{(w, wd) : w \in D^*,\ d \in D\}, \\ r^* &= \{(wd_1, \ldots, wd_k) : w \in D^*,\ (d_1, \ldots, d_k) \in r\}, \\ \mathit{clone} &= \{wdd : w \in D^*,\ d \in D\}. \end{aligned}$$

So $\mathcal{N}^*$ is a tree where every node has a rank equal to the size of D; indeed a node $w \in D^*$ has wd as successors for every $d \in D$. The relation *child* gives the successors of a node. The relations r^* induce additional dependencies between siblings that come from the initial structure. The unary relation *clone* is a curious predicate saying that a node is exactly the same as its parent. This predicate is very useful in encoding of other structures into iterated structures.

Example: Consider a two element structure $\mathcal{N}_2 = \langle \{0, 1\}, \mathit{left} \rangle$ with one unary relation that holds only for 0. The structure $\mathcal{N}_2^*$ has elements $\{0, 1\}^*$. The relation $\mathit{left}^*(w)$ holds if w ends with 0. Hence $\mathcal{N}_2^*$ is the full binary tree with left^* designating left sons. The *clone* relation is not important here: it holds in a left child whose parent is also a left child (or analogously for right).

Example: Consider the set of natural numbers with the standard ordering: $\mathcal{N}_\leq = \langle \mathbb{N}, \leq \rangle$. In this case $\mathcal{N}_\leq^*$ is a tree of infinite branching where siblings are linearly ordered. Moreover, the *clone* predicate designates a node whose position between its siblings is the same as that of the father. The structure is shown in Figure 7, where the circled nodes satisfy the clone predicate. The clone predicate allows us to define in this tree paths representing a possible behavior of one register with $+1$ and -1 operations: paths of the form $i_1 i_2, \ldots$ with $i_1 = 0$ and $|i_{k+1} - i_k| \leq 1$. The idea is presented in Figure 7. The clone predicate allows us to go one step to the left or one step to the right in the horizontal linear order when going one level down in the tree.

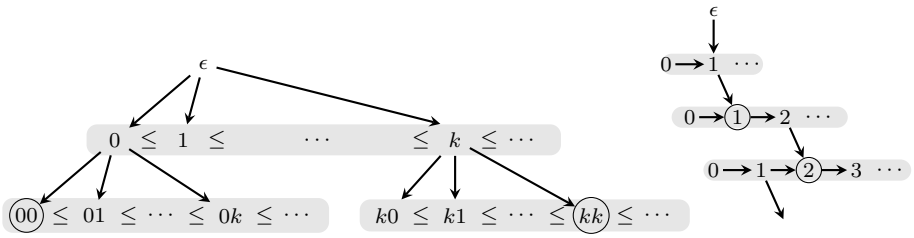

Fig. 7 Structure $\mathcal{N}_\leq^*$, and a representation of the sequence $1, 2, 1, \ldots$ in $\mathcal{N}_\leq^*$

In general, for a given structure $\mathcal{N} = \langle D, r_1, \ldots \rangle$, the structure $\mathcal{N}^*$ can be seen as a generalization of the notion of the full binary tree. Given a finite alphabet Σ, a Σ-labeling of $\mathcal{N}$ is just a function $t : \mathcal{N}^* \to \Sigma$. We are going to define a notion of automaton that accepts sets of labeled iterated structures.

First, let $\mathcal{MF}(\Sigma, Q)$ be the set of monadic second-order formulas with free variables $\{X_{clone}\} \cup \{X_q : q \in Q\}$ over the signature of $\mathcal{N}$ enriched with monadic predicates P_a for $a \in \Sigma$. The automaton is:

$$\mathcal{A} = \langle Q, \Sigma, q^0 \in Q, \delta : Q \to \mathcal{MF}(\Sigma, Q), \Omega : Q \to \mathbb{N} \rangle$$

The *acceptance* of $t : \mathcal{N}^* \to \Sigma$ by $\mathcal{A}$ is defined in terms of the *acceptance game* $\mathcal{G}(t, \mathcal{A})$.

- The positions for Eve are $D^* \times Q$, the positions for Adam are $D^* \times (Q \to \mathcal{P}(D))$. The initial position is (ε, q^0). Recall that D^* is the set of elements of $\mathcal{N}^*$, in particular ε is the root of $\mathcal{N}^*$.
- In a position (w, q) Eve chooses a function $f : Q \to \mathcal{P}(D)$ so that $\mathcal{N}, \mathcal{V} \vDash \delta(q)$ where $\mathcal{V}(X_q) = f(q)$ and $\mathcal{V}(X_{clone})$ is the set consisting of the last element of w, or $\varnothing$ if w has length ≤ 1. The game moves to the position (w, f).
- In a position (w, f) Adam chooses $q' \in Q$ and $d' \in f(q')$. The game moves to (wd', q').
- The rank of a position (w, q) is $\Omega(q)$. All the positions for Adam have rank 0.

A labeled iterated structure is *accepted* if Eve has a winning strategy in this game from the initial position. Hence an automaton accepts a set of labeled iterated structures, for a fixed initial structure. The following theorem has been formulated by A. Muchnik. The sketch of his proof can be found in [110]. The result is proved with a different proof method in [124].

Theorem 24 (Transfer theorem [110, 124]). *Let $\mathcal{N}$ be a relational structure. Automata over labeled $\mathcal{N}$-iterated structures are closed under boolean operations and projection. If the MSOL theory of $\mathcal{N}$ is decidable then the emptiness of automata over labeled $\mathcal{N}$-iterated structures is decidable.*

In particular, when $\mathcal{N}_2$ is the two element structure from the example above, Theorem 24 gives decidability of the MSOL theory of the binary tree. Observe that modal automata (cf. Section 3.2) are a special case of automata from this section for the structures of empty signature. We have seen that modal automata and the mu-calculus are essentially the same. Following the same ideas one can construct mu-calculi for different kinds of relations on the set of successor states.

Structure iteration is a powerful operation preserving decidability of MSOL theories: that is transforming a structure with decidable MSOL theory into another structure with decidable MSOL theory. Among others, it is an entry point to the so-called pushdown hierarchy, model-checking higher-order

pushdown systems, and higher-order recursive schemes. This is a vast subject that would require a chapter on its own [74, 116, 31, 93, 75, 30].

4.3.2 Guarded logics

The idea of guarded logics comes from looking at the translation of the modal logic into first-order logic. In this translation the boolean connectives are translated to themselves, and the modalities are translated as follows:

$$\begin{array}{rcl} [\langle a\rangle\alpha]^*(x) & \text{is} & \exists y.\ R_a(x,y) \wedge [\alpha]^*(y) \\ [[a]\alpha]^*(x) & \text{is} & \forall y.\ R_a(x,y) \Rightarrow [\alpha]^*(y)\ . \end{array}$$

As for MSOL, the translation is parameterized by a free variable intended to stand for the current state.

The image of this translation is called the *modal fragment* of first-order logic. Recall that the satisfiability problem for first-order logic over transition systems is undecidable. But it is decidable for the modal fragment since it is decidable for modal logic. This immediately brings on the question: what makes the modal fragment so special, and can it be extended? The idea behind guarded fragment is to provide an answer to this question by focusing on the restricted use of quantifiers.

We say that a quantification is *guarded* if it is of the form

$$\exists y.\gamma(x,y) \wedge \psi(x,y) \quad \text{or} \quad \forall y.\gamma(x,y) \Rightarrow \psi(x,y),$$

where $\gamma(x,y)$ is $R_b(x,y)$ or $R_b(y,x)$ for some action b, and $\psi(x,y)$ is a formula whose free variables are at most x and y. The name "guarded" comes from the fact that the quantified variable has to be related to the free variable by the transition relation. For example the formula

$$\forall y.R_a(x,y) \Rightarrow x = y$$

says that all transitions on a are self-loops. The syntax also permits to talk directly about the reverse of transitions.

The *guarded fragment* [5] is the set of formulas of first order logic that contains atomic formulas, and is closed under boolean operations and guarded quantification. The presentation here is limited to the simplest variant. The fragment gets even more interesting for signatures with relations of higher arity.

Since the guarded fragment inherits many good properties of modal logic, its extension with fixpoints should inherit those of the mu-calculus. This is indeed the case. If one takes some care as to how fixpoints are applied, one can even recover many good properties of the mu-calculus.

Let W be a unary relation variable, let $\psi(W,x)$ be a guarded formula whose free variables are as displayed, and where W appears only positively

and not in guards. Then we can build formulas:

$$[\mathsf{lfp}\ Wx.\ \psi](x) \quad \text{and} \quad [\mathsf{gfp}\ Wx.\ \psi](x)\ .$$

The semantics is as expected: The formula $[\mathsf{lfp}\ Wx.\psi](s)$ is true in a transition system $\mathcal{M}$ iff s is in the least fixpoint of the operator mapping a set of states S' to $\{s' : \mathcal{M} \vDash \psi(S', s')\}$, i.e., to the set of states s' that make $\psi(W, x)$ satisfied when W is interpreted by S'. The extension of the guarded logic with these two constructs is called *guarded fixpoint logic*.

Let us see an example formula of guarded fixpoint logic that is not equivalent to a mu-calculus formula:

$$\big(\exists xy.\ R_b(x,y)\big) \wedge \big(\forall xy.\ R_b(x,y) \Rightarrow \exists z.\ R_b(y,z)\big) \wedge \\ \forall xy.\ R_b(x,y) \Rightarrow [\mathsf{lfp}\ Wz.\ \forall y.\ R_b(y,z) \Rightarrow W(y))](x)\ .$$

The first two conjuncts say that there is a transition labeled by b, and that every such transition can be extended to an infinite path. The third conjunct says that every source of a b-transition has only finitely many predecessors on b transitions. Thus the formula implies that there is an infinite forward chain of b-transitions but no infinite backward chain. This example shows the use of backwards modalities and the price to pay for them: we can write formulas having only infinite models.

Despite this observation the satisfiability problem for the guarded fragment extended with fixpoints is decidable, and the complexity is the same as for the mu-calculus.

Theorem 25 (Guarded fixpoint logic [61]). *The satisfiability problem for guarded fixpoint logic over transition systems is* EXPTIME*-complete.*

Adaptations of many results presented in this chapter hold for guarded fixpoint logic: game characterization of model-checking, a tree model property, bisimulation invariance, iteration of structures [57, 58, 21].

5 Related work

As with any good concept, the mu-calculus can be approached from many directions. The first point of view is to consider it as a logic of programs. The family of logics of programs is divided into two groups. In exogenous logics, a program is a part of a formula; in endogenous logics, a program is a part of a model. Dynamic logic and Hoare logic are examples of exogenous logics. Temporal logic, and Floyd diagrams are endogenous logics. The mu-calculus also belongs to this second group.

Exogenous logics merit a small digression. Among them, dynamic logics are the closest to the mu-calculus. Historically, the research on dynamic

logics has been an intermediate step to major results of the theory of the mu-calculus [111, 112]. Dynamic logics were developed independently by Slawicki [102], and Pratt [102]. A propositional version of dynamic logic has been proposed and studied by Fisher and Ladner [50]. A survey of Harel gives a very good overview of the subject [62]. A more recent reference is a book of Harel, Kozen and Tiuryn [63].

The second point of view is that the mu-calculus is a propositional version of the least fixpoint logic: an extension of first-order logic with fixpoint operators. From this point of view Y. Moschovakis' work in model theory have laid foundations for the logic [86]. Least fixpoint logic and the closely related inflationary fixpoint logic are intensively studied in finite-model theory [41, 67, 78, 59]. Inflationary fixpoint logic has its propositional version too [40]; while it has greater expressive power, it is less algorithmically manageable than the mu-calculus.

The third point of view is to consider the mu-calculus as a basic modal logic extended with a fixpoint operator. Modal logic was proposed by philosophers in the beginning of the 20th century [19]. In the 1950s a possible word semantics was introduced, and since then modal logic has proven to be an appealing language to describe properties of transition systems. A number of ways of using a fixpoint operator in program logics have been proposed [106, 65, 95, 98]. The mu-calculus as it is known now was formulated by Kozen in [76].

The expressive completeness theorem for the mu-calculus, Theorem 22, is an analog of van Benthem's theorem [12] saying that a first-order formula is equivalent to a modal formula if and only if it is bisimulation invariant. So Theorem 22 says that mu-calculus is to MSOL what modal-logic is to first-order logic. Expressive completeness results have been also proved for extensions of the mu-calculus as well as for its fragments [58, 68].

As stated in Theorem 8, the model-checking problem is equivalent to checking emptiness of nondeterministic parity automata on infinite trees. Similarly, the satisfiability problem is linked to automata emptiness, cf. Theorem 14. Many arguments in this chapter have an automata-theoretic flavor. This explains the relevance of the study of determinization and complementation operations for automata for the theory of the mu-calculus [101, 119, 105, 96, 35].

As with any successful formalism it is very tempting to extend the mu-calculus while retaining most of its good properties. In some sense the expressive completeness theorem tells us that it is not possible to keep all the good properties. In Section 4.3.2 we have described guarded fixpoint logics. The mu-calculi with backwards modalities, loop modalities etc. have been studied separately [118, 9, 26]. There exist also a mu-calculus for timed transition systems [27, 64]. Quantitative versions of the mu-calculus have also been proposed: be it for probabilistic transition systems [82, 1, 56, 85], or for some form of discounting [49]. In Section 4.3.1 we have seen how to extend mu-calculus to the case when the set of successors of a state has some structure, as for example linear order. Another extension in a similar spirit is coalgebraic mu-calculus [121].

The results on the model-checking problem have been discussed in Section 2.4. It is worth mentioning that the problem has also been studied on some special classes of transition systems: bounded tree and clique-width [92], bounded entanglement [14], undirected graphs [16, 38].

The alternation hierarchy discussed in Section 3.4 is the most established way of stratifying mu-calculus properties. Another hierarchy is the variable hierarchy obtained by limiting the number of variables that can be used to write a formula. Most common program logics: CTL, PDL, CTL*, and even the game logic of Parikh [94] are contained in the first two levels of this hierarchy. Still the variable hierarchy is strict [15].

As we have seen in Section 4.2 most program logics like CTL* or PDL can be translated into the mu-calculus. It would be interesting to understand which formulas of the mu-calculus correspond to formulas of, say, CTL. Put it differently, for a logic L we would like to decide if a given mu-calculus formula is equivalent to a formula of L. Decidability of this problem is known for several fragments of first-order logic [24, 25, 11, 97, 22, 23]. Yet, the problem is open for all major program logics like CTL, CTL*, or PDL.

There exist two other surveys on the subject that present the logic from different angles [29, 120]. For coverage in depth of the theory of the mu-calculus we refer the reader to the book of Arnold and Niwiński [8].

References

1. de Alfaro, L., Majumdar, R.: Quantitative solution of omega-regular games. J. Comput. Syst. Sci. **68**(2), 374–397 (2004)
2. Alur, R., Henzinger, T., Kupferman, O.: Alternating-time temporal logic. In: FOCS'97, pp. 100–109 (1997)
3. Andersen, H.: Model checking boolean graphs. TCS **126**(1), 3–30 (1994)
4. Andersen, H.R.: Partial model checking. In: LICS'95, pp. 398–407 (1995)
5. Andréka, H., van Benthem, J., Neméti, I.: Modal logics and bounded fragments of predicate logic. Journal of Philosophical Logic **27**, 217–274 (1998)
6. Arnold, A.: The mu-calculus alternation-depth hierarchy is strict on binary trees. RAIRO–Theoretical Informatics and Applications **33**, 329–339 (1999)
7. Arnold, A., Crubille, P.: A linear time algorithm to sove fixpoint equations on transition systems. Inf. Process. Lett. **29**(57-66) (1988)
8. Arnold, A., Niwiński, D.: Rudiments of μ-calculus. Elsevier (2001)
9. Arnold, A., Walukiewicz, I.: Nondeterministic controllers of nondeterministic processes. In: J. Flum, E. Grädel, T. Wilke (eds.) Logic and Automata, *Texts in Logic and Games*, vol. 2, pp. 29–52. Amsterdam University Press (2007)
10. Bekic, H.: Definable operation in general algebras, and the theory of automata and flowcharts. In: C.B. Jones (ed.) Programming Languages and Their Definition - Hans Bekic (1936-1982), *LNCS*, vol. 177, pp. 30–55. Springer (1984)
11. Benedikt, M., Segoufin, L.: Regular tree languages definable in FO and in FO$_{mod}$. ACM Trans. Comput. Log. **11**(1) (2009)
12. Benthem, J.: Modal Logic and Classical Logic. Bibliopolis, Napoli (1983)
13. Berwanger, D.: Game logic is strong enough for parity games. Studia Logica **75**(2), 205–219 (2003)

14. Berwanger, D., Grädel, E., Kaiser, L., Rabinovich, R.: Entanglement and the complexity of directed graphs. Theor. Comput. Sci. **463**, 2–25 (2012)
15. Berwanger, D., Grädel, E., Lenzi, G.: The variable hierarchy of the mu-calculus is strict. Theory Comput. Syst. **40**(4), 437–466 (2007)
16. Berwanger, D., Serre, O.: Parity games on undirected graphs. Inf. Process. Lett. **112**(23), 928–932 (2012)
17. Bhat, G., Cleaveland, R.: Efficient model checking via the equational μ-calculus. In: LICS, pp. 304–312 (1996)
18. Björklund, H., Vorobyov, S.G.: A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. Discrete Applied Mathematics **155**(2), 210–229 (2007)
19. Blackburn, R., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press (2001)
20. Bloem, R., Chatterjee, K., Jobstmann, B.: Games and synthesis. In: Hanbook of Model-Checking. Springer (2014)
21. Blumensath, A., Kreutzer, S.: An extension to Muchnik's theorem. Journal of Logic and Computation **13**, 59–74 (2005)
22. Bojanczyk, M., Segoufin, L., Straubing, H.: Piecewise testable tree languages. Logical Methods in Computer Science **8**(3) (2012)
23. Bojanczyk, M., Straubing, H., Walukiewicz, I.: Wreath products of forest algebras, with applications to tree logics. Logical Methods in Computer Science **8**(3) (2012)
24. Bojanczyk, M., Walukiewicz, I.: Characterizing EF and EX tree logics. Theoretical Computer Science **358**(2-3), 255–272 (2006)
25. Bojanczyk, M., Walukiewicz, I.: Forest algebras. In: J. Flum, E. Grädel, T. Wilke (eds.) Logic and Automata, *Texts in Logic and Games*, vol. 2, pp. 107–132. Amsterdam University Press (2007)
26. Bonatti, P.A., Lutz, C., Murano, A., Vardi, M.Y.: The complexity of enriched mu-calculi. Logical Methods in Computer Science **4**(3) (2008)
27. Bouyer, P., Cassez, F., Laroussinie, F.: Timed modal logics for real-time systems: Specification, verification and control. Journal of Logic, Language and Information **20**, 169–203 (2011)
28. Bradfield, J.: The modal mu-calculus alternation hierarchy is strict. Theoretical Computer Science **195**, 133–153 (1997)
29. Bradfield, J., Stirling, C.: Modal mu-calculi. In: P. Blackburn, J. van Benthem, F. Wolter (eds.) The Handbook of Modal Logic, pp. 721–756. Elesvier (2006)
30. Broadbent, C., Carayol, A., Ong, L., Serre, O.: Recursion schemes and logical reflection. In: LICS, pp. 120–129 (2010)
31. Carayol, A., Wöhrle, S.: The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In: FSTTCS'03, *Lecture Notes in Computer Science*, vol. 2914, pp. 112–124 (2003)
32. Chatterjee, K., Henzinger, M.: An $O(n^2)$ time algorithm for alternating Büchi games. In: SODA, pp. 1386–1399. SIAM (2012)
33. Church, A.: Applications of recursive arithmetic to the problem of cricuit synthesis. In: Summaries of the Summer Institute of Symbolic Logic, vol. I, pp. 3–50. Cornell Univ., Ithaca, N.Y. (1957)
34. Cleaveland, R., Steffen, B.: A linear model checking algorithm for the alternation-free modal μ-caluclus. Formal Methods in System Design **2**, 121–147 (1993)
35. Colcombet, T., Zdanowski, K.: A tight lower bound for determinization of transition labeled Büchi automata. In: ICALP (2), *LNCS*, vol. 5556, pp. 151–162 (2009)
36. Condon, A.: The complexity of stochastic games. Inf. Comput. **96**(2), 203–224 (1992)
37. D'Agostino, G., Hollenberg, M.: Logical questions concerning the mu-calculus: Interpolation, Lyndon and Loś-Tarski. J. Symb. Log. **65**(1), 310–332 (2000)
38. D'Agostino, G., Lenzi, G.: On modal mu-calculus over reflexive symmetric graphs. J. Log. Comput. **23**(3), 445–455 (2013)

39. Dam, M.: CTL* and ECTL* as a fragments of the modal μ-calculus. Theor. Comput. Sci. **126**(1), 77–96 (1994)
40. Dawar, A., Grädel, E., Kreutzer, S.: Inflationary fixed points in modal logic. ACM Trans. Comput. Log. **5**(2), 282–315 (2004)
41. Ebbinghaus, H.D., Flum, J.: Finite model theory, 2nd edn. Perspectives in Mathematical Logic. Springer (1999)
42. Ebbinghaus, H.D., Flum, J., Thomas, W.: Mathematical Logic. New York: Springer-Verlag (1984)
43. Emerson, E., Jutla, C., Sistla, A.: On model-checking for the mu-calculus and its fragments. Theoretical Computer Science **258**(1-2), 491–522 (2001)
44. Emerson, E.A.: Temporal and modal logic. In: J. Leeuven (ed.) Handbook of Theoretical Computer Science Vol.B, pp. 995–1072. Elsevier (1990)
45. Emerson, E.A., Jutla, C.S.: The complexity of tree automata and logics of programs. In: Foundations of Computer Science, pp. 328–337 (1988)
46. Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy. In: Proc. FOCS'91, pp. 368–377 (1991)
47. Emerson, E.A., Jutla, C.S.: The complexity of tree automata and logics of programs. SIAM J. Comput. **29**(1), 132–158 (1999)
48. Emerson, E.A., Lei, C.: Efficient model checking in fragments of propositional mu-calculus. In: First IEEE Symp. on Logic in Computer Science, pp. 267–278 (1986)
49. Fischer, D., Grädel, E., Kaiser, L.: Model checking games for the quantitative mu-calculus. Theor. Comput. Syst. **47**, 696–719 (2010)
50. Fisher, M., Ladner, R.: Propositional modal logic of programs. In: 9th ACM Ann. Aymp. on Theory of Computing, pp. 286–294 (1977)
51. Fisher, M., Ladner, R.: Propositional dynamic logic of regular programs. Journal of Computer and System Sciences **18**, 194–211 (1979)
52. Friedmann, O.: An exponential lower bound for the latest deterministic strategy iteration algorithms. Logical Methods in Computer Science **7**(3) (2011)
53. Friedmann, O., Hansen, T.D., Zwick, U.: A subexponential lower bound for the random facet algorithm for parity games. In: SODA, pp. 202–216. SIAM (2011)
54. Friedmann, O., Lange, M.: The modal mu-calculus caught off guard. In: TABLEAUX, *LNCS*, vol. 6793, pp. 149–163 (2011)
55. Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In: Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01), *LNCS*, vol. 2102, pp. 53–65. Springer (2001)
56. Gimbert, H., Zielonka, W.: Perfect information stochastic priority games. In: ICALP, *LNCS*, vol. 4596, pp. 850–861 (2007)
57. Grädel, E.: Guarded fixed point logics and the monadic theory of countable trees. Theor. Comput. Sci. **288**(1), 129–152 (2002)
58. Grädel, E., Hirsch, C., Otto, M.: Back and forth between guarded and modal logics. ACM Trans. Comput. Log. **3**(3), 418–463 (2002)
59. Grädel, E., Kolaitis, P., Libkin, L., Marx, M., Spencer, J., Vardi, M., Venema, Y., Weinstein, S.: Finite Model Theory and Its Applications. EATCS Series: Texts in Theoretical Computer Science. Springer-Verlag (2007)
60. Grädel, E., Thomas, W., T.Wilke (eds.): Automata, Logics, and Infinite Games, *Lecture Notes in Computer Science*, vol. 2500. Springer-Verlag (2002)
61. Grädel, E., Walukiewicz, I.: Guarded fixpoint logic. In: LICS'99, pp. 45–55 (1999)
62. Harel, D.: Dynamic logic. In: Handbook of Philosophical Logic Vol II, pp. 497–604. D.Reidel Publishing Company (1984)
63. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press (2000)
64. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. Inf. Comput. **111**(2), 193–244 (1994)
65. Hitchcock, P., Park, D.: Induction rules and termination proofs. In: M. Nivat (ed.) 1st Internat. Coll. on Autoamta, Languages and Programming, pp. 225–251 (1973)

66. Hoffman, A., Karp, R.: On nonterminating stochastic games. Management Science **12**, 359–370 (1966)
67. Immerman, N.: Descriptive Complexity. Graduate texts in computer science. Springer (1999)
68. Janin, D., Lenzi, G.: On the rlationship between monadic and weak monadic second order logic on arbitrary trees, with applications to the mu-calculus. Fundam. Inform. **61**(3-4), 247–265 (2004)
69. Janin, D., Walukiewicz, I.: Automata for the mu-calculus and related results. In: MFCS '95, *LNCS*, vol. 969, pp. 552–562 (1995)
70. Janin, D., Walukiewicz, I.: On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In: CONCUR'96, *LNCS*, vol. 1119, pp. 263–277 (1996)
71. Jurdziński, M.: Deciding the winner in parity games is in UP∩co-UP. Information Processing Letters **68**(3), 119–124 (1998)
72. Jurdziński, M.: Small progress measures for solving parity games. In: STACS, *LNCS*, vol. 1770, pp. 290–301 (2000)
73. Jurdzinski, M., Paterson, M., Zwick, U.: A deterministic subexponential algorithm for solving parity games. SIAM J. Comput. **38**(4), 1519–1532 (2008)
74. Knapik, T., Niwinski, D., Urzyczyn, P.: Higher-order pushdown trees are easy. In: FoSSaCS' 02, *LNCS*, vol. 2303, pp. 205–222 (2002)
75. Kobayashi, N.: Types and higher-order recursion schemes for verification of higher-order programs. In: POPL'09, pp. 416–428. ACM (2009)
76. Kozen, D.: Results on the propositional mu-calculus. Theoretical Computer Science **27**, 333–354 (1983)
77. Kupferman, O.: Automata on infinite objects. In: Hanbook of Model-Checking. Springer (2014)
78. Libkin, L.: Elements of Finite Model Theory. Springer (2004)
79. Long, D.E., Browne, A., Clarke, E.M., Jha, S., Marrero, W.R.: An improved algorithm for the evaluation of fixpoint expressions. In: CAV'94, *LNCS*, vol. 818, pp. 338–350 (1994)
80. Maksimova, L.L.: Absence of interpolation and of Beth's property in temporal logics with "the next" operation. Siberian Mathematical Journal **32**(6), 109–113 (1991)
81. Martin, D.: Borel determinacy. Ann. Math. **102**, 363–371 (1975)
82. McIver, A., Morgan, C.: Results on the quantitative - calculus qM. ACM Trans. Comput. Log. **8**(1) (2007)
83. Mcmillan, K.: Interpolation: Proofs in the service of model checking. In: Hanbook of Model-Checking. Springer (2014)
84. McNaughton, R.: Infinite games played on finite graphs. Ann. Pure and Applied Logic **65**, 149–184 (1993)
85. Mio, M.: Game semantics for probabilistic mu-calculi. Ph.D. thesis, University of Edinburgh (2012)
86. Moschovakis, Y.: Elementary Induction on Abstract Structures. North-Holand (1974)
87. Moss, L.S.: Coalgebraic logic. Annals of Pure and Applied Logic **96**, 277–317 (1999). Erratum published APAL 99:241–259, 1999
88. Mostowski, A.W.: Regular expressions for infinite trees and a standard form of automata. In: Fifth Symposium on Computation Theory, *LNCS*, vol. 208, pp. 157–168 (1984)
89. Mostowski, A.W.: Games with forbidden positions. Tech. Rep. 78, University of Gdansk (1991)
90. Muller, D., Schupp, P.: Alternating automata on infinite trees. Theoretical Computer Science **54**, 267–276 (1987)
91. Niwiński, D.: Fixed points vs. infinte generation. In: Logic in Computer Science, pp. 402–409 (1988)
92. Obdrzálek, J.: Clique-width and parity games. In: CSL, *LNCS*, vol. 4646, pp. 54–68 (2007)

93. Ong, C.H.L.: On model-checking trees generated by higher-order recursion schemes. In: LICS'06, pp. 81–90 (2006)
94. Parikh, R.: The logic of games and its applications. Annals of Discrete Mathematics **24**, 111–140 (1985)
95. Park, D.: Finiteness is μ-ineffable. Theoretical Computer Science **3**, 173–181 (1976)
96. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. Logical Methods in Computer Science **3**(3) (2007)
97. Place, T., Segoufin, L.: A decidable characterization of locally testable tree languages. Logical Methods in Computer Science **7**(4) (2011)
98. Pratt, V.: A decidable μ-calculus: preliminary report. In: Proc. 22nd Ann IEEE Symp. on Foundations of Computer Science, pp. 421–427 (1981)
99. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Transactions of the AMS **141**, 1–23 (1969)
100. Rabin, M.O.: Weakly definable relations and special automata. Mathematical Logic and Foundations of Set Theory pp. 1–23 (1970)
101. Safra, S.: On the complexity of ω-automata. In: 29th IEEE Symp. on Foundations of Computer Science (1988)
102. Salwicki, A.: Formalized algorithmic languages. Bull. Acad. Polon. Sci., Ser. Sci. Math. Astron. Phys. **18**, 227–232 (1970)
103. Schewe, S.: Solving parity games in big steps. In: FSTTCS, *LNCS*, vol. 485, pp. 449–460 (2007)
104. Schewe, S.: An optimal strategy improvement algorithm for solving parity and payoff games. In: CSL, *LNCS*, vol. 5213, pp. 369–384 (2008)
105. Schewe, S.: Büchi complementation made tight. In: STACS, *LIPIcs*, vol. 3, pp. 661–672 (2009)
106. Scott, D., de Bakker, J.: A theory of programs (1969). Unpublished notes, IBM, Vienna, 1969
107. Seidl, H.: Deciding equivalence of finite tree automata. SIAM Journal of Computing **19**, 424–437 (1990)
108. Seidl, H.: Fast and simple nested fixpoints. Information Processing Letters **59**(6), 303–308 (1996)
109. Seidl, H., Neumann, A.: On guarding nested fixpoints. In: CSL, *LNCS*, vol. 1683, pp. 484–498 (1999)
110. Semenov, A.: Decidability of monadic theories. In: MFCS'84, *LNCS*, vol. 176, pp. 162–175. Springer-Verlag (1984)
111. Streett, R.S.: Propositional dynamic logic of looping and converse is elementarily decidable. Information and Control **54**, 121–141 (1982)
112. Streett, R.S., Emerson, E.A.: The propositional mu-calculus is elementary. In: ICALP, *Lecture Notes in Computer Science*, vol. 172, pp. 465–472 (1984)
113. Streett, R.S., Emerson, E.A.: An automata theoretic procedure for the propositional mu-calculus. Information and Computation **81**, 249–264 (1989)
114. Thomas, W.: Languages, automata, and logic. In: G. Rozenberg, A. Salomaa (eds.) Handbook of Formal Language Theory, vol. III, pp. 389–455. Springer (1997)
115. Thomas, W.: Infinite games and verification. In: Computer Aided Verification (CAV'02), *LNCS*, vol. 2404, pp. 58–64 (2002)
116. Thomas, W.: Constructing infinite graphs with a decidable MSO-theory. In: MFCS, vol. 2747, pp. 113–124. LNCS (2003)
117. Thomas, W.: Church's problem and a tour through automata theory. In: A. Avron, N. Dershowitz, A. Rabinovich (eds.) Pillars of Computer Science, *LNCS*, vol. 4800, pp. 635–655 (2008)
118. Vardi, M.: Reasoning about the past with two-way automata. In: Automata, Languages and Programming ICALP 98, *LNCS*, vol. 1443, pp. 628–641 (1998)
119. Vardi, M.Y.: The Büchi complementation saga. In: STACS, no. 4393 in LNCS, pp. 12–22 (2007)

120. Vardi, M.Y., Wilke, T.: Automata: From logics to algorithms. In: J. Flum, E. Grädel, T. Wilke (eds.) Logic and Automata, *Texts in Logic and Games*, vol. 2. Amsterdam University Press (2007)
121. Venema, Y.: Automata and fixed point logic: A coalgebraic perspective. Inf. Comput. **204**(4), 637–678 (2006)
122. Vöge, J., Jurdziński, M.: A discrete strategy improvement algorithm for solving parity games (Extended abstract). In: CAV, *LNCS*, vol. 1855, pp. 202–215 (2000)
123. Walukiewicz, I.: Completeness of Kozen's axiomatisation of the propositional μ-calculus. Information and Computation **157**, 142–182 (2000)
124. Walukiewicz, I.: Monadic second order logic on tree-like structures. Theoretical Computer Science **257**(1–2), 311–346 (2002)
125. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science **200**, 135–183 (1998)

Index