TASK QUARTERLY 9 No 3, 293–316

THE MODAL μ-CALCULUS: A SURVEY

GIACOMO LENZI

University of Pisa,
Department of Mathematics,
Largo Pontecorvo 5, 56127 Pisa, Italy
lenzi@mail.dm.unipi.it

(Received 14 September 2004)

Abstract: The modal μ-calculus is an extension of modal logic with two operators μ and ν, which give the least and greatest fixpoints of monotone operators on powersets. This powerful logic is widely used in computer science, in the area of verification of correctness of concurrent systems. In this survey we review both the theoretical aspects of the modal μ-calculus and its applications to computer science.

Keywords: μ-calculus, fixed points, model checking

1. Introduction

The subject of this survey is the *modal μ-calculus*, a formalism introduced by Dexter Kozen in [1], which nowadays represents a very active research area in both theoretical and practical computer science. The letter μ reminds that the μ-calculus is a logic capable of expressing least and greatest solutions of fixpoint equations $X = f(X)$, where f is a monotone function mapping some powerset into itself. In fact, the notation $\mu X.f(X)$ is used for the least fixpoint of the function f, and the notation $\nu X.f(X)$ is used for the greatest fixpoint of f. As the name suggests, modal μ-calculus is built on top of modal logic; for many other μ-calculi see [2].

Despite the intense research activity which has been done in the modal μ-calculus in its two decades of life, not very much is known on the modal μ-calculus as a mathematical object. This is true for example about the expressiveness of the logic: it is not completely clear what we can say with a formula of modal μ-calculus, although the beautiful Janin-Walukiewicz Theorem, see [3], sheds light on this issue. Surely, the modal μ-calculus subsumes many traditional logics of programs, hence it captures the most used correctness properties of computer systems. But what is there besides the traditional logics of programs? This is an informal but interesting open problem.

Another, more applicative issue is the *model checking problem*, namely whether it is possible, and how, to check efficiently (that is, in polynomial time) whether a finite structure satisfies a formula of modal μ-calculus. This is a formal problem corresponding to the informal problem of deciding whether a computer system

(assumed to have finitely many states) is correct with respect to its specifications. Since the model checking problem is in the complexity class $NP \cap co-NP$ by [4], there is an evident link with the famous problem $P =? NP$.

This survey has no claim of originality or exhaustivity, and is just an updated collage of previous surveys, in particular [5–7].

The rest of the paper is organized as follows. In Section 2 we give some applicative motivations. In Sections 3 and 4 we review the syntax and semantics of the modal μ-calculus. In Section 5 give some examples of formulas. In Section 6 we recall some logics of programs which can be translated into the modal μ-calculus. In Section 7 we develop some theoretical aspects. Finally in Section 8 we discuss the model checking problem, which is the most interesting applicative aspect of the modal μ-calculus.

2. The applicative context

The most important application area for the modal μ-calculus is the formal specification and verification of computer systems. In fact the production of any computer system, be it software or hardware, must start with the *specification* of what the system to be produced is supposed to do.

However the production of computer systems, like all human activities, is error-prone, so it may happen that the final product does not meet the initial specification, and this may cause an incorrect behavior of the system. Another source of problems is that the specification itself may be imprecise or incomplete, so that the system may satisfy the specification but yet behave incorrectly.

So, in order to ensure a correct behavior of the system, one has to work in two directions:

1. make the specifications precise and understandable;
2. make it feasible to verify whether a system meets a specification.

It is commonly accepted that these goals are easier to accomplish if one uses *formal methods*. In the formal approach, a system is modeled by some mathematical structure, and a specification is a list of mathematical properties which may be true or false in the structure.

One widely used structure for modeling computer systems is represented by *transition systems*, which we will discuss in a subsequent section. The most interesting properties which one wants to verify depend on the kind of application. For example, an operating system installed on a computer is supposed to provide a permanent service to the user of the computer, so an interesting property could be:

The system never stops (unless the computer is switched off).

This is an example of a *safety* property. Intuitively, safety properties say that something bad will never happen.

As another example, it may be that some resource, say a printer, is shared by many users, and the program wich monitors may have to fulfill the requests of each user to access the resource. If the resource is mutually exclusive, in general not all the requests can be fulfilled immediately, as the resource may be already occupied when

a request arrives; so the monitor must schedule the allocation of the resource in a way that the following property holds:

Every request for the resource is eventually acknowledged.

This is an example of a *liveness* property. Intuitively, liveness properties say that something good will eventually happen. Hence, in a sense, liveness is dual to safety.

Note that often, safety properties alone or liveness properties alone are trivial to satisfy, and the problem is to satisfy properties of both kinds simultaneously.

As a third example, consider a message buffer. Usually it is assumed that the buffer has a First In First Out behavior, namely the buffer preserves the order of the messages; this can be expressed by the property:

If a message m enters the buffer before a message m', then m exits the buffer before m'.

Properties of this kind are called *precedence* properties. They express that things happen in the right order. They can be viewed as generalizations of safety or liveness properties.

Properties like in the examples above can be formalized either directly in a temporal logic, or indirectly in the modal μ-calculus, via a translation of temporal logic into the modal μ-calculus.

3. The syntax of the modal μ-calculus

3.1. The syntax of formulas

Fix a set Rel of relation symbols, a set At of atoms and a set Var of variables. The *formulas* of the modal μ-calculus over Rel, At and Var are defined by the following syntax:

$$\phi ::= \mathit{false} \mid \mathit{true} \mid X \mid P \mid \neg P \mid \phi \vee \phi \mid \phi \wedge \phi \mid \langle R \rangle \phi \mid [R]\phi \mid \mu X.\phi \mid \nu X.\phi,$$

where $X \in Var$, $P \in At$ and $R \in Rel$.

The operators $\neg, \vee, \wedge$ are the usual boolean connectives. The operators $[R]$ and $\langle R \rangle$ are the box and diamond operators of modal logic. The operators μ and ν give the least and greatest fixpoint operators.

Note that negation is allowed only in front of atoms. In this way, the function defined by any formula $\phi(X)$ containing a variable X is semantically monotone in X, hence the least and greatest fixpoints of the function exist.

However, the negation of *closed* formulas can be defined by exchanging P with $\neg P$, $\wedge$ with $\vee$, $\langle R \rangle$ with $[R]$ and μ with ν.

3.2. Simultaneous fixpoints

There is a variant of the modal μ-calculus that admits simultaneous fixpoints of several formulas. This does not increase the expressive power, but often allows more modular and easier to read formalizations. The mechanism for building simultaneous fixpoints is the following: given formulas $\phi_1, \ldots, \phi_k$ and variables $X_1, \ldots, X_k$,

$$S := \begin{cases} X_1 = \phi_1 \\ \vdots \\ X_k = \phi_k \end{cases}$$

is called a *system*, which can be used to build the formulas $\mu X_i.S$ and $\nu X_i.S$.

4. The semantics of the modal μ-calculus

4.1. Transition systems

Fix a set Rel of relation symbols, a set At of atoms and a set Var of variables. A *transition system* or *Kripke structure* for Rel, At and Var is a structure $\mathcal{K}$ with universe V (whose elements are called vertexes or states), binary relations $E_R \subseteq V \times V$ for each $R \in Rel$ (whose elements are called edges or arcs or transitions), and sets $I_P \subseteq V$ for every $P \in At$ and $I_X \subseteq V$ for every $X \in Var$.

4.2. Kripke semantics

The formulas of the modal μ-calculus are evaluated on transition systems at a particular vertex. Given a sentence ϕ and a transition system $\mathcal{K}$ with vertex v, we write $\mathcal{K}, v \models \phi$ to denote that ϕ holds in $\mathcal{K}$ at vertex v. The set of vertexes $v \in V$ such that $\mathcal{K}, v \models \phi$ is denoted by $\|\phi\|^{\mathcal{K}}$. We omit the definition of $\|\phi\|^{\mathcal{K}}$ for the obvious cases. For the modal operators:

$\|\langle R\rangle\phi\|^{\mathcal{K}} := \{v : \textit{ there is } w \textit{ such that } (v,w) \in E_R \textit{ and } w \in \|\phi\|^{\mathcal{K}}\}$;
$\|[R]\phi\|^{\mathcal{K}} := \{v : \textit{ for all } w, \textit{ if } (v,w) \in E_R, \textit{ then } w \in \|\phi\|^{\mathcal{K}}\}$.

To understand the semantics of fixed point formulas, note that a formula $\phi(X)$ with a variable X defines on every transition system $\mathcal{K}$ (with vertex set V, and with interpretations for variables other than X occurring in ϕ) an operator $\phi^{\mathcal{K}} : P(V) \to P(V)$ assigning to every set $X \subseteq V$ the set $\phi^{\mathcal{K}}(X) := \|\phi\|^{\mathcal{K},X} = \{v \in V : (\mathcal{K}, X), v \models \phi\}$. As X occurs only positively in ϕ, the operator $\phi^{\mathcal{K}}$ is *monotone* for every $\mathcal{K}$, and therefore, by a well-known theorem due to Knaster and Tarski, has a least fixpoint, $\mathit{lfp}(\phi^{\mathcal{K}})$, and a greatest fixpoint, $\mathit{gfp}(\phi^{\mathcal{K}})$. Now we put:

$$\|\mu X.\phi\|^{\mathcal{K}} := \mathit{lfp}(\phi^{\mathcal{K}}), \quad \|\nu X.\phi\|^{\mathcal{K}} := \mathit{gfp}(\phi^{\mathcal{K}}).$$

4.3. Simultaneous fixpoints

Given a system

$$S := \begin{cases} X_1 = \phi_1 \\ \vdots \\ X_k = \phi_k \end{cases}$$

of k equations and a Kripke structure $\mathcal{K}$, we have an operator $S^{\mathcal{K}}$ mapping a k-tuple $\overline{X} = (X_1, \ldots, X_k)$ of sets of vertexes to $S_1^{\mathcal{K}}(\overline{X}), \ldots, S_k^{\mathcal{K}}(\overline{X})$ with

$$S_i^{\mathcal{K}}(\overline{X}) = \|\phi_i\|^{(\mathcal{K},\overline{X})}.$$

As $S^{\mathcal{K}}$ is monotone, there exist the least fixpoint $\mathit{lfp}(S) = (X_1^{\mu}, \ldots, X_k^{\mu})$ and the greatest fixpoint $\mathit{gfp}(S) = (X_1^{\nu}, \ldots, X_k^{\nu})$.

Now set $\|\mu X_i.S\|^{\mathcal{K}} := X_i^{\mu}$ and $\|\nu X_i.S\|^{\mathcal{K}} := X_i^{\nu}$.

Simultaneous fixpoints can always be eliminated, in fact:

Proposition 1. (see [2]) *Every formula of the modal μ-calculus with simultaneous fixpoints can be translated into an equivalent formula without simultaneous fixpoints.*

4.4. Approximants

Another way of defining the semantics of μ-calculus is via approximants. The idea is to approximate least fixpoints from below and greatest fixpoints from above. We present approximants as follows.

Consider *infinitary modal logic*, which is defined by the syntax:

$$\phi ::= true \mid false \mid X \mid P \mid \neg P \mid \bigvee_{i\in I} \phi_i \mid \bigwedge_{i\in I} \phi_i \mid \langle R\rangle\phi \mid [R]\phi,$$

where I ranges over all possible sets of indexes, even infinite.

The semantics of infinitary modal logic can be defined on transition systems by extending the semantics of modal logic in the obvious way.

Let α be an ordinal and let $\phi(X)$ be a formula of infinitary modal logic. The *approximant formulas* $\mu^\alpha X.\phi(X)$ and $\nu^\alpha X.\phi(X)$ are formulas of infinitary modal logic inductively defined by:

- $\mu^0 X.\phi(X) = false$;
- $\mu^{\alpha+1} X.\phi(X) = \phi(\mu^\alpha X.\phi(X))$;
- $\mu^\lambda X.\phi(X) = \bigvee_{\alpha<\lambda} \mu^\alpha X.\phi(X)$ for λ limit ordinal;
- $\nu^0 X.\phi(X) = true$;
- $\nu^{\alpha+1} X.\phi(X) = \phi(\nu^\alpha X.\phi(X))$;
- $\nu^\lambda X.\phi(X) = \bigwedge_{\alpha<\lambda} \nu^\alpha X.\phi(X)$ for λ limit ordinal.

Now it results:

Theorem 1. (Knaster-Tarski)

- $\|\mu X.\phi(X)\|^{\mathcal{K}} = \|\mu^{|\mathcal{K}|'} X.\phi(X)\|^{\mathcal{K}}$;
- $\|\nu X.\phi(X)\|^{\mathcal{K}} = \|\nu^{|\mathcal{K}|'} X.\phi(X)\|^{\mathcal{K}}$,

where $|\mathcal{K}|' = |\mathcal{K}|$ *if* $\mathcal{K}$ *is finite, and* $|\mathcal{K}|' = |\mathcal{K}|^+$ *otherwise.*

4.5. Bisimulation

As a modal logic, the modal μ-calculus distinguishes between transition systems only up to behavioral equivalence, captured by the notion of bisimulation.

A *bisimulation* between two transition systems $\mathcal{K}$ and $\mathcal{K}'$ is a relation $Z \subseteq V \times V'$ between the domains of $\mathcal{K}$ and $\mathcal{K}'$, respecting atoms and variables in the sense that $\mathcal{K}, v \models P$ iff $\mathcal{K}', v' \models P$ for $P \in At$ and $(v, v') \in Z$, and similarly for variables, and satisfying the following back and forth conditions.

Forth: for all $(v, v') \in Z$, $R \in Rel$ and every w such that $(v, w) \in E_R$, there exists a w' such that $(v', w') \in E'_R$ and $(w, w') \in Z$.

Back: for all $(v, v') \in Z$, $R \in Rel$ and every w' such that $(v', w') \in E'_R$, there exists a w such that $(v, w) \in E_R$ and $(w, w') \in Z$.

Two transition systems with a distinguished point, $\mathcal{K}, u$ and $\mathcal{K}', u'$, are called bisimilar if there is a bisimulation Z between $\mathcal{K}$ and $\mathcal{K}'$ such that $(u, u') \in Z$.

5. Examples of formulas

First of all, a natural question is: *what do we gain in adding fixpoints to modal logic?*. The answer is: *we pass from "local" to "global" properties.* In fact, informally speaking, one can see that a formula of modal logic, say of length n, only can talk about the points of a structure which have distance at most n from the current point; instead, as we will see in this section, a formula of μ-calculus can talk about the entire structure. Thus the gain in expressiveness is enormous.

Another question could be: *what is the difference between μ and ν?*. The answer is: *μ corresponds to finite computations, and ν corresponds to infinite computations*. This will be clarified by some examples.

We have seen that two kinds of desirable properties of systems are safety and liveness. In terms of modal μ-calculus, it is not unreasonable to say that μ is liveness and ν is safety. Consider first simple ν formulas. For example:

$$\nu Z.P \wedge [R]Z$$

is a relativized always formula: P is true along every R-path. Slightly more complex is the weak until formula

$$\nu Z.Q \vee (P \wedge [R]Z)$$

saying that on every R-path, P holds until Q holds (in particular, P holds forever if Q never holds in the path). Both formulas can be understood directly via the fixpoint construction, or via the idea of ν as infinite looping: for example the second formula is true if either Q holds, or if P holds and wherever we go next the formula is true, *etc.*, and because the fixpoint is maximal, we can repeat forever.

In contrast, formulas beginning with μ require something to happen, and thus are liveness properties. For example:

$$\mu Z.P \vee [R]Z$$

says that on all infinite paths, P eventually holds; and

$$\mu Z.Q \vee (P \wedge [R]Z)$$

is a strong until operator: on all paths, P holds until Q holds, and Q does eventually hold. Again, these can be understood by regarding μ as a finite looping: in the second case, we are no longer allowed to repeat the unfolding forever, so we must eventually bottom out in the Q disjunct.

All these formulas can be translated into the temporal logic *CTL*. A formula which goes beyond *CTL* is:

$$\nu X.P \wedge [R][R]X,$$

which says that P holds at every even position. This kind of cyclic properties is not expressible in temporal logic, but it is expressible in *PDL*. But in order to exploit fully the expressive power of the modal μ-calculus, we must mix fixpoints that depend on one another. Consider the formula:

$$\mu Y.\nu Z.(P \wedge [R]Y) \vee (\neg P \wedge [R]Z).$$

This formula looks horrible, but it has a simple meaning, which can be seen using the slogans. $\mu Y \ldots$ is true iff $\nu Z \ldots$ is true iff $(P \wedge [R]Y) \vee (\neg P \wedge [R]Z)$, which is true if either P holds and at the next vertexes we loop back to $\mu Y \ldots$, or P fails and at the next vertexes we loop back to $\nu Z \ldots$. By the slogan that μ means finiteness, we can only loop $\mu Y \ldots$ finitely many times on any path, hence P is true only finitely often on any path.

It is also worth considering the negative of this, namely infinitely often, as it illustrates the finite unfolding of μ within an outer ν. Consider the formula

$$\nu Y.\mu Z.\langle R\rangle Y \vee \langle S\rangle Z.$$

Applying the slogans, we see that this is true if either we can do an R and loop to Y, or do an S and loop to Z. We can only loop to Z finitely often, so we can only

do finitely many consecutive Ss; but if we do an R and loop back out to Y, we then re-enter the inner fixpoint, free to do another independent finitely many Ss.

We shall see in a later section that this so-called alternation of fixpoint operators does indeed give ever more expressive power as the number of alternations increases. It also appears to increase the complexity of model checking: all known algorithms are exponential in the alternation, but whether this is necessarily the case is the main remaining open problem about the modal μ-calculus. However, the practitioner who is alarmed by the thought of even more complex formulas can take comfort in the widely asserted proposition that one never actually needs more than two alternating fixpoints, and even two is a bit unusual. In fact, many would go further and say that in real life we are only interested in safety properties.

There are two main reasons why alternating formulas might appear. The first is if one has a front end working in CTL^* (or another complex logic) and is translating into modal μ-calculus for the main engine. This is theoretically possible, by work of Mads Dam [8], but the translation is sufficiently complex that it would be surprising to find a tool doing this. The second is if one needs to express fairness properties. In fact, both weak fairness (an event continuously enabled must happen) and strong fairness (an event enabled infinitely often must happen) can be written in the modal μ-calculus, and require mixing fixpoints. For example, to say that a relation R is fairly treated means that there are no paths on which R is enabled infinitely often, but occurs only finitely often; this can be written as:

$$\nu X.\mu Y.\nu Z.[R]X \wedge (\langle R\rangle true \Rightarrow [-R]Y) \wedge [-R]Z,$$

where $[-R]\phi$ means $\bigwedge_{S\neq R}[S]\phi$.

Of course, we may be in a situation where the system is inherently unfair – *e.g.*, it is a normal interleaving of concurrent components – and in that case we can use a similar idea to express properties such as: P eventually holds on all paths, provided that R is fairly treated, which is left as an exercise to the reader. However, fairness can also be handled by other means, such as defining the set of fair paths outside the logic, and adapting algorithms to apply only to fair paths.

6. Some sublogics

6.1. LTL

Linear time temporal logic (LTL) has been the first temporal logic used in verification, see [9]. This logic is obtained from propositional logic by adding the operator O (next) and the operator U (until). The syntax of LTL is given by:

$$\phi ::= P \mid \neg\phi \mid \phi \wedge \phi \mid O\phi \mid \phi U \phi,$$

where P denotes an atomic proposition.

The formulas of LTL are interpreted only on the "linear" Kripke structure $(\omega, Succ)$, where ω is the set of all natural numbers, and $Succ$ is the relation which relates each $n \in \omega$ to its successor $n+1$. Intuitively, the natural numbers represent instants of time. $O\phi$ means that ϕ holds at the next time, and $\phi U \psi$ means that there is a time where ψ is true, and before the first such time, ϕ always holds.

Two useful derived operators are $F\phi = trueU\phi$ and $G\phi = \neg F\neg\phi$. Thus, $F\phi$ means that ϕ is true sometimes, and $G\phi$ means that ϕ is true always. Another

derived operator is the before operator B, given by $\phi B\psi = \neg((\neg\phi)U\psi)$. That is, if ψ ever occurs, then there is a ϕ occurring before (the formula is trivially true if ψ never occurs). This operator is useful to express precedence properties.

The logic LTL can be translated into the modal μ-calculus via the translation t defined by:

- $t(P) = P$;
- t commutes with the booleans;
- $t(O\phi) = \langle Succ\rangle t(\phi)$;
- $t(\phi U\psi) = \mu X.t(\psi) \vee (t(\phi) \wedge \langle Succ\rangle X)$.

6.2. CTL

Computation tree logic (CTL) is a branching time temporal logic, introduced in [10]. Unlike LTL, in CTL the evolution of time is seen as nondeterministic, and every instant of time has several successors, rather than exactly one as in LTL; so, the entire structure of time is a tree. In CTL we are allowed to use, besides the LTL operators O and U, also the path quantifiers E (for some path) and A (for all paths). More precisely, the syntax of CTL is defined by:

$$\phi ::= P \mid \neg\phi \mid \phi \wedge \phi \mid QO\phi \mid Q(\phi U\phi),$$

where Q is a path quantifier, that is, E or A.

As usual, we will interpret the logic CTL only over structures which are total, in the sense that every point has at least one successor. This implies that from every point x of the structure there is some infinite path starting from x. Now, $EO\phi$ means that there is an infinite path, starting from the current point, where ϕ holds the next time, and this turns out to be equivalent to simply $\langle R\rangle\phi$. Likewise, $AO\phi$ is equivalent to $[R]\phi$. Moreover, $E(\phi U\psi)$ means that there is an infinite path, starting from the current point, where $\phi U\psi$ holds, and $A(\phi U\psi)$ means that for every infinite path starting from the current point, $\phi U\psi$ holds.

Note that also in CTL we can use the derived operators F, G, B introduced in LTL; in particular, $EF\phi$ will mean that ϕ is true sometimes, and $AG\phi$ will mean that ϕ is true always (but, unlike the LTL case, this will be true of arbitrary total trees, rather than just linear structures).

We can translate CTL into the modal μ-calculus as follows:

- $t(EO\phi) = \langle R\rangle t(\phi)$;
- $t(AO\phi) = [R]t(\phi)$;
- $t(E(\phi U\psi)) = \mu X.t(\psi) \vee (t(\phi) \wedge \langle R\rangle X)$;
- $t(A(\phi U\psi)) = \mu X.t(\psi) \vee (t(\phi) \wedge [R]X)$.

6.3. CTL*

We have seen that CTL introduces linear time operators and path quantifiers. However, not all combinations of these operators are allowed in CTL. If we allow any such combination, we get the stronger logic CTL^*, see [11]. Often the syntax of CTL^*

is presented by talking about state formulas and path formulas. Here instead, we present the following, equivalent but simpler (in our opinion) syntax:

$$\phi ::= P \mid \neg\phi \mid \phi \wedge \phi \mid E\lambda(\phi_1, \ldots, \phi_n),$$

where $\lambda(P_1, \ldots, P_n)$ is a formula of *LTL*. (We extend from *CTL* to *CTL** the proviso that the models should be total).

Clearly, to translate *CTL** into the modal μ-calculus, it is enough to translate formulas of the form $E\lambda(P_1, \ldots, P_n)$. A way to do this is to express $E\lambda(P_1, \ldots, P_n)$ in monadic second order logic, observe that the formula is invariant under bisimulation, and use the Janin-Walukiewicz theorem to conclude that the formula is expressible in the modal μ-calculus. More direct translations have been given, see [8]; however, *any* such translation must be exponentially complicated, because the modal μ-calculus is decidable in exponential time, whereas *CTL** is decidable in no less than double exponential time.

6.4. PDL

Propositional Dynamic Logic (*PDL*, see [12]) is another, widely used logic for programs, which can be seen as a generalization of Hoare logic. The idea is to view a program as a relation between its inputs and its outputs, hence for every program we will have a modality; so, for example, a partial correctness assertion about a program p will have the form $P \to [p]Q$, whereas a total correctness will have the form $P \to \langle p\rangle Q$. The logic *PDL* allows us to express correctness assertions and quite a lot more.

In principle, one can consider programs of any programming language; here we consider only programs built from regular expressions, that is:

$$p ::= a \mid p;p \mid p \cup p \mid p^*,$$

where a denotes an atomic program, $p;p$ denotes concatenation, $p \cup p$ denotes nondeterministic choice, and p^* denotes finite iteration.

The syntax of *PDL* formulas now is:

$$\phi ::= P \mid \neg\phi \mid \phi \wedge \phi \mid \langle p\rangle\phi.$$

We can translate *PDL* into the modal μ-calculus as follows:

- $t(\langle a\rangle\phi) = \langle a\rangle t(\phi)$;
- $t(\langle p;p'\rangle\phi) = t(\langle p\rangle t(\langle p'\rangle\phi))$;
- $t(\langle p \cup p'\rangle\phi) = t(\langle p\rangle\phi) \vee t(\langle p'\rangle\phi)$;
- $t(\langle p^*\rangle\phi) = \mu X.t(\phi) \vee t(\langle p\rangle X)$.

7. Some theory

7.1. Fixpoint regeneration and the fundamental semantic theorem

Assume a structure $\mathcal{K}$ and a formula ϕ. Suppose that we annotate the vertexes with sets of subformulas, such that the sets are *locally consistent*: that is, v is annotated with P iff v models P, and similarly for $\neg P$; v is annotated with a conjunction iff it is annotated with both conjuncts; v is annotated with a disjunction iff it is annotated with at least one disjunct; if v is annotated with $[R]\psi$ (respectively, $\langle R\rangle\psi$), then every (respectively, at least one) R-successor is annotated with ψ; if v is

annotated with a fixpoint or fixpoint variable, it is annotated with the body of the fixpoint. We write $v@\psi$ if the vertex v is annotated with ψ.

We call such an annotated structure a *quasi-model*.

A *choice function* f is a function which, for every disjunctive formula $\psi_1 \vee \psi_2$ and every vertex v annotated with $\psi_1 \vee \psi_2$, chooses one disjunct $f(v, \psi_1 \vee \psi_2)$; and for every subformula $\langle R\rangle\psi$, and every vertex v annotated with $\langle R\rangle\psi$, chooses one R-successor $w = f(v, \langle R\rangle\psi)$ annotated with ψ.

A *pre-model* is a quasi-model equipped with a choice function.

Given a pre-model with choice function f, the *dependencies* of a vertex v that satisfies a formula ψ are defined thus: $v@\psi_1 \wedge \psi_2 > v@\psi_i$ for $i = 1, 2$; $v@[R]\psi > w@\psi$ for every pair $(v, w) \in E_R$; $v@\psi_1 \vee \psi_2 > v@f(v, \psi_1 \vee \psi_2)$; $v@\langle R\rangle\psi > f(v, \langle R\rangle\psi)@\psi$; $v@\mu Z.\psi > v@\psi$, and similarly for ν; $v@Z > s@\psi$, where Z is bound by $\mu Z.\psi$ or $\nu Z.\psi$. A *trail* is a maximal chain of dependencies.

If every trail has the property that the highest (*i.e.*, with the outermost binding fixpoint) variable occurring infinitely often is a ν-variable, the pre-model is said to be *well-founded*. Equivalently: in any trail, a μ-variable can only occur finitely often unless a higher variable is encountered.

The fundamental theorem on the semantics of the modal μ-calculus can now be stated:

Theorem 2. *A well-founded pre-model is a model: in a well-founded pre-model, if v is annotated with ψ, then indeed $v \models \psi$.*

The theorem in this form is due to [13], from which the term "well-founded pre-model" is taken. Stirling and Walker [14] present a tableau system for model-checking on finite structures, and the soundness theorem for this is essentially a finite version of the fundamental theorem with a more relaxed notion of choice; the later infinite-state version of [15, 16] is the fundamental theorem, again with a slight relaxation on choice.

A converse is also true:

Theorem 3. *If in some structure $v \models \phi$, then there is a locally consistent annotation of the structure and a choice function which make the structure a well-founded pre-model.*

The fundamental theorem, in its various guises, is the precise statement of the slogan that "μ means finiteness". To explain why it is true, we need to make a finer analysis of approximants.

Assume a structure $\mathcal{K}$ and a formula ϕ. Let $Y_1, \ldots, Y_n$ be the μ-variables of ϕ, in an order compatible with formula inclusion: that is, if $\mu Y_j.\psi_j$ is a subformula of $\mu Y_i.\psi_i$, then $i \leq j$. If Y_i is some inner fixpoint, then its denotation depends on the meaning of the fixpoints enclosing it: for example, in the formula $\mu Y_1.\langle R\rangle\mu Y_2.(P \vee Y_1) \vee \langle S\rangle Y_2$, to calculate the inner fixpoint μY_2 we need to know the denotation of Y_1. We may ask: what is the least approximant of Y_1 that could be plugged in to make the formula true? Having fixed that, we can then ask what approximant of Y_2 is required. The idea is the notion of signature. A *signature* is a sequence $\sigma = \alpha_1, \ldots, \alpha_n$ of ordinals, such that the i^{th} least fixpoint will be interpreted by its $\alpha_i{}^{\text{th}}$ approximant (calculated relative to the outer approximants).

The definition and use of signatures inevitably involves some slightly irritating book-keeping, and they appear in several forms in the literature. In [13], the Fischer-Ladner closure of ϕ was used, rather than the set of subformulas. The signatures were defined by syntactically unfolding fixpoints, rather than by semantic approximants. In [14] and following work, a notion of constant was used, which allows some of the book-keeping to be moved into the logic. Although all the notions and proofs using them are interconvertible, the constant variant is perhaps easier to follow.

Add to the language a countable set of *constants* $U,V,\ldots$ Constants will be defined to stand for maximal fixpoints or approximations of minimal fixpoints. Specifically, given a formula ϕ, let $Y_1,\ldots,Y_n$ be the μ-variables as above, let $Z_1,\ldots,Z_m$ be the ν-variables, let $\sigma=\alpha_1,\ldots,\alpha_n$ be a signature, and let $U_1,\ldots,U_n,V_1,\ldots,V_m$ be constants, which will be associated with the corresponding variables. They are given semantics thus: if Y_i is bound by $\mu Y_i.\psi_i$, then $\|U_i\|\sigma$ is $\|\mu^{\alpha_i}Y_i.\psi_i'\|\sigma$, where ψ_i' is obtained from ψ_i by substituting the corresponding constants for the free fixpoint variables of $\mu Y_i.\psi_i$. If Z_i is bound by $\nu Z_i.\psi_i$, its semantics is $\|\nu Z_i.\psi_i'\|\sigma$. Given an arbitrary subformula ψ of ϕ, we say a vertex v satisfies ψ with signature σ, written $v\models_\sigma\psi$, if $v\in\|\psi'\|\sigma$, where ψ' is ψ with its free fixpoint variables substituted by the corresponding constants.

Order signatures lexicographically. Now, given a pre-model for ϕ, extend the annotations so that each subformula at v is accompanied by a signature – write $v@\psi[\sigma]$. Such an extended annotation is said to be *locally consistent* if the signature is unchanged or decreases by passing through boolean, modal or ν-variable dependencies, and when passing through $v@Y_i$ it strictly decreases in the i^{th} component and is unchanged in the $1,\ldots,(i-1)^{\text{th}}$ components. It can now be shown, by a slightly delicate but not too difficult induction, that if $v@\psi[\sigma]$, then $v\models_\sigma\psi$. Furthermore, given a well-founded pre-model, one can construct a locally consistent signature annotation – essentially, the Y_i component of σ in $v@\psi[\sigma]$ is the maximum number (in the transfinite sense) of Y_i occurrences without meeting a higher variable in trails from $v@\psi$, and so on; the well-foundedness of the pre-model guarantees that this is well-founded. This gives the fundamental theorem. (A little care is required to get the details of this argument correct, as will be seen from an inspection of the proofs in [13–15].)

The converse is quite easy: given a model, annotate the vertexes by the subformulas they satisfy; for $v@\psi$ assign the least σ such that $v\models_\sigma\psi$; and choose a choice function that always chooses the successor with least signature. It is easy to show that this is a well-founded pre-model and signature assignment.

7.2. The finite model property and decidability

The notion of well-founded pre-model was introduced by Streett and Emerson in order to establish the following theorem.

Theorem 4. *The modal μ-calculus is decidable; that is, it is decidable whether a formula has some model.*

As a corollary of this proof, they obtain the small model property:

Theorem 5. *If a modal μ-calculus formula has a model, then it has a finite model, of size exponential in the size of the formula.*

These results were obtained by appealing to automata theory. First of all, one can easily show that if a formula has a model, it has a model that is a tree (by unraveling) and has bounded branching degree (by cutting off all the branches that are not actually required by some diamond subformula, so that no more branches are left than the number of diamonds of the formula). We can now construct an automaton that accepts such bounded-branching tree models, by combining a finite-state automaton to check the local consistency (*i.e.*, to check that the putative model is a pre-model), and a Rabin automaton (see Subsection 7.6) to check that the pre-model is well-founded. Thus the formula is satisfiable if this product automaton accepts some tree. Now established automata theory tells us that: (a) this question is decidable, (b) if such an automaton accepts some tree, it accepts a regular tree, that is, one that is the unraveling of a finite system; this gives the results.

One can, however, obtain a small model property, and thence decidability, by a more direct argument on models. Take a pruned tree model as above, and equip it with an annotation and choice function to make it a well-founded pre-model. Consider an outermost maximal fixpoint $\nu Z.\psi$, say. Look at the first occurrence of a vertex v labelled by Z; let F be the set of formulas annotating v. Now, if there is a vertex w below v that is also annotated by F, prune the tree at w, and identify w with v. Repeat for all Z labelled vertexes with repetitions below them. On the remaining infinite branches of the tree, Z does not occur infinitely often; and by the well-foundedness of the pre-model, no immediately inferior least fixpoint occurs infinitely often either; so we still have a well-founded pre-model. Now repeat the procedure with the next maximal fixpoint in, and so on. When all fixpoints have been processed, the result is a finite graph that is still a well-founded pre-model. The repetition detection is reminiscent of the classical filtration technique for *PDL*. Unfortunately, filtration is not sound for minimal fixpoints, so we have this less constructive procedure. It generates, naively, a model of triply exponential size; but it is intuitively easy, and of course the existence of even a triply exponential model is enough to give decidability. One can improve the complexity somewhat by being less simple-minded, but to obtain the exponential upper bound of [13], which is in fact also the lower bound, one needs the more sophisticated techniques of automata or games (for more on automata and games, see the Subsections 7.6 and 7.7).

A different model for obtaining the small model property proceeds via a normal form result for the modal μ-calculus. A formula is an *automaton normal form*, anf, if it belongs to the following sublogic:

- P, $\neg P$ and Z are anfs;
- if ϕ_1 and ϕ_2 are anfs, so is $\phi_1 \vee \phi_2$;
- if ϕ is an anf, then so are $\mu Z.\phi$ and $\nu Z.\phi$;
- If each Γ_i is a finite set of anfs, $R_i \neq R_j$ when $i \neq j$, and Σ is a finite set of atoms and their negations, then:

$$\Sigma \wedge \mathit{Cover}_{R_1}\Gamma_1 \wedge \ldots \wedge \mathit{Cover}_{R_n}\Gamma_n$$

is an anf, where we introduce the notation:

$$\mathit{Cover}_R\Gamma = (\bigwedge_{\phi\in\Gamma} \langle R\rangle\phi) \wedge ([R]\bigvee_{\phi\in\Gamma} \phi).$$

The term *Cover* suggests that the formulas belonging to Γ "cover" the R-successors of the current point in a nontrivial way.

Walukiewicz in [17] (see also [18]) proves that every (closed) modal μ-calculus formula is equivalent to an anf formula. Walukiewicz calls anfs "disjunctive formulas". However they are very close to characteristic automata (and in fact games). The proof of the normal form result uses automata. What is interesting is that an anf formula is satisfiable iff, by replacing all minimal fixpoints $\mu X.\phi(X)$ with $\phi(false)$ and all maximal fixpoints $\nu X.\phi(X)$ with $\phi(true)$, we get a satisfiable formula. Note that the resulting formula is a modal μ-calculus formula without fixpoints, for which proof of the finite model property is easy.

7.3. Axiomatization

A related problem to decidability is the question of providing an axiomatization of the theory of the modal μ-calculus. In his original paper, Kozen presents the following axiomatization of the equational theory, where $\phi \leq \psi$ means $\phi \vee \psi = \psi$ (that is, ϕ implies ψ), taking $\langle R\rangle$ and μ as primitives and defining $[R]$ and ν by duality:

1. axioms for boolean algebras;
2. $\langle R\rangle\phi \vee \langle R\rangle\psi = \langle R\rangle(\phi \vee \psi)$;
3. $\langle R\rangle\phi \wedge [R]\psi \leq \langle R\rangle(\phi \wedge \psi)$;
4. $\langle R\rangle false = false$;
5. $\phi(\mu X.\phi(X)) \leq \mu X.\phi(X)$;
6. if $\phi(\psi) \leq \psi$ then $\mu X.\phi(X) \leq \psi$.

Axiom 5 is the axiom of fixed point induction, in dual form; rule 6 says that μ is indeed the least pre-fixed point. Note that for monotonic functions, the least fixpoint and the least pre-fixed point coincide. (An equivalent presentation of the axiom system is as an extension of minimal multi-modal logic K, with the additional axiom $\phi(\mu X.\phi(X)) \Rightarrow \mu X.\phi(X)$ and the additional inference rule: if $\phi(\psi) \Rightarrow \psi$ then $\mu X.\phi(X) \Rightarrow \psi$.)

However, despite the naturalness of this axiomatization, Kozen was unable to show that it was complete. He was, however, able to show completeness for a restricted language, the language of aconjunctive formulas, in which (roughly) fixpoint variables are not allowed to occur in both branches of a conjunction.

Completeness for the full language remained open for more than a decade, until it was finally solved by Walukiewicz in [17], who established that Kozen's axiomatization is indeed complete. The proof is very involved and utilises the automata normal forms described above (which generalize the aconjunctive fragment). It is reasonably straightforward to show using tableaux that if an anf formula ϕ is consistent (that is, if $\phi \Rightarrow false$ is not derivable in the system) then it has a model. Much harder to prove is that every (closed) formula is provably equivalent within the axiom system to an anf formula. Walukiewicz uses automata and games to show this. More information can also be found in the notes [19].

It must be said that in Walukiewicz's proof, the transitivity of the implication is implicitly assumed. This is a form of cut, so we are left with the problem of finding a cut-free proof system for the modal μ-calculus. Such systems exist for modal logic,

but are not available (to the author's knowledge) even for much weaker logics than the modal μ-calculus, like *PDL*.

7.4. Alternation depth

We now look at the definition of *(fixpoint) alternation depth*. The idea is to count alternations of minimal and maximal fixpoint operators, but to do so in a way that only counts real dependency. The paradigm is always eventually versus infinitely often: the always eventually formula

$$\nu Y.(\mu Z.P \vee \langle R\rangle Z) \wedge \langle R\rangle Y$$

is, using brute-force model checking, really no worse to compute than two disjoint fixpoints, since the inner fixpoint can be computed once and for all, rather than separately on each outer approximant; on the other hand, the infinitely often formula

$$\nu Y.\mu Z.(P \vee \langle R\rangle Z) \wedge \langle R\rangle Y$$

really does need the full double induction on approximants.

The definition of Emerson and Lei takes care of this by observing that the eventually subformula is a closed subformula, and giving a definition that ignores closed subformulas when counting alternations. The stronger notion of Niwiński, which also has the advantage of being robust under translation to modal equation systems, also observes that, for example, $\mu X.\nu Y.[R]Y \wedge \mu Z.[R](X \vee Z)$ although it looks like a $\mu/\nu/\mu$ formula, is morally a μ/ν formula, since the inner fixpoint does not refer to the middle fixpoint.

It is possible to give algorithms that compute the alternation depth of a formula, and this is how the notion was presented by Emerson and Lei in [20]. However, for our purposes it is easier to start from a definition of classes of formulas, formalizing the idea of a $\mu/\nu/\mu$ formula, etc.; such a definition is analogous to the usual definition of quantifier alternation for predicate logic, an analogy which will be exploited later. This is how Niwiński [21] presents the notion of alternation, and we follow his presentation.

A formula ϕ is said to be in the classes $\Sigma_0^{N\mu}$ and $\Pi_0^{N\mu}$ iff it contains no fixpoint operators. To form the class $\Sigma_{n+1}^{N\mu}$ ($\Pi_{n+1}^{N\mu}$, respectively), take $\Sigma_n^{N\mu} \cup \Pi_n^{N\mu}$ and close under the following rules:

1. if $\phi,\phi_2 \in \Sigma_{n+1}^{N\mu}$ ($\Pi_{n+1}^{N\mu}$, respectively), then $\phi_1 \vee \phi_2$, $\phi_1 \wedge \phi_2$, $\langle R\rangle\phi_1$, $[R]\phi_1 \in \Sigma_{n+1}^{N\mu}$ ($\Pi_{n+1}^{N\mu}$, respectively);
2. if $\phi \in \Sigma_{n+1}^{N\mu}$ ($\Pi_{n+1}^{N\mu}$, respectively), then $\mu Z.\phi \in \Sigma_{n+1}^{N\mu}$ ($\nu Z.\phi \in \Pi_{n+1}^{N\mu}$, respectively);
3. if $\phi(Z),\psi \in \Sigma_{n+1}^{N\mu}$ ($\Pi_{n+1}^{N\mu}$, respectively), then $\phi(\psi) \in \Sigma_{n+1}^{N\mu}$ ($\Pi_{n+1}^{N\mu}$, respectively), provided that no free variable of ψ is captured by a fixpoint operator in ϕ.

If we omit the last clause, we get the definition of simple-minded alternation $\Sigma_n^{S\mu}$, that just counts syntactic alternation; if we modify the last clause to read: provided that ψ is a closed formula, then we obtain the Emerson-Lei notion $\Sigma_n^{EL\mu}$. (We write just Σ_n^{μ} when the distinction is not important, or when we are making a statement that applies to all versions.)

To get the symmetrical notion of alternation depth of ϕ, we can define it to be the last n such that $\phi \in \Sigma^{\mu}_{n+1} \cap \Pi^{\mu}_{n+1}$, and we denote it by $ad(\phi)$. To make these definitions clear, consider the following examples:

- The always eventually formula is $\Pi^{S\mu}_2$, but not $\Sigma^{S\mu}_2$, and so its simple alternation depth is 2. However, in the Emerson-Lei notion, it is also $\Sigma^{EL\mu}_2$, since $\nu Y.W \wedge \langle R\rangle Y$ is $\Pi^{EL\mu}_1$ and so $\Sigma^{EL\mu}_2$, and by substituting the closed $\Sigma^{EL\mu}_2$ (and in fact $\Sigma^{EL\mu}_1$) formula $\mu Z.P \vee \langle R\rangle Z$ for W we get always eventually in $\Sigma^{EL\mu}_2$; hence the Emerson-Lei (and Niwiński) alternation depth is 1.
- The infinitely often formula is Σ^{μ}_2 but not Π^{μ}_2, in all three definitions, and so has alternation depth 2.
- The formula $\mu X.\nu Y.[R]Y \wedge \mu Z.[R](X \vee Z)$ is $\Sigma^{S\mu}_3$, but not $\Pi^{S\mu}_3$; it is also $\Sigma^{EL\mu}_3$ but not $\Pi^{EL\mu}_3$, since there are no closed subformulas to bring the substitution clause into play. However, in the Niwiński definition, it is actually $\Sigma^{N\mu}_2$: $\nu Y.[R]Y \wedge W$ is $\Pi^{N\mu}_1$ and so $\Sigma^{N\mu}_2$; we can substitute the $\Sigma^{N\mu}_1$ formula $\mu Z.[R](X \vee Z)$ for W without variable capture, and so $\nu Y.[R]Y \wedge \mu Z.[R](X \vee Z)$ is $\Sigma^{N\mu}_2$; and now we can add the outer fixpoint, still remaining in $\Sigma^{N\mu}_2$.

Alternation depth plays an important role in model checking. A natural question is whether the hierarchy of properties definable by Σ^{μ}_n formulas is actually a strict hierarchy, or whether it collapses at some point such that no further alternation is needed. This problem remained open for a while; by 1990, it was known that $\Sigma^{N\mu}_2 \neq \Pi^{N\mu}_2$ (see [22]). No further advance was made until 1996, when the strictness of the hierarchy was established by Bradfield (see [23]).

Theorem 6. *For every n, there is a formula $\phi \in \Sigma^{\mu}_n$ which is not equivalent to any Π^{μ}_n formula.*

Bradfield established this for $\Sigma^{N\mu}_n$, which implies the result for the other two notions. At the same time, [24] independently established a slightly weaker hierarchy theorem for $\Sigma^{EL\mu}_n$.

The proof of [24] is technically complex, and the underlying stratagem is not easy. Bradfield's proof appears technically complex, but most of the complexity is really just routine recursion-theoretic coding; the underlying stratagem is quite simple, and in some ways surprising. If one takes first-order arithmetic, one can add fixpoint operators to it, and one can then define a fixpoint alternation hierarchy in arithmetic. A standard coding and diagonalization argument shows that this hierarchy is strict, see [25]. The trick now is to transfer this hierarchy to the modal μ-calculus. Simply by writing down the semantics, it is clear that if one takes a recursively presented transition system and codes it into the integers, then for a modal formula $\phi \in \Sigma^{\mu}_n$, its denotation $\|\phi\|$ is describable by an arithmetic Σ^{μ}_n formula. However, it is also possible, given any arithmetic fixpoint formula χ, to build a transition system and a modal formula ϕ, of the same alternation depth as χ, such that $\|\phi\|$ is characterized by χ. If we take χ to be a strict Σ^{μ}_n arithmetic formula, then no Π^{μ}_n arithmetic formula is equivalent to it, and therefore no Π^{μ}_n modal formula is equivalent to ϕ. The transition system that is constructed is infinite, but by the finite model property, the hierarchy transfers down to the class of finite models.

Both proof techniques construct explicit examples of hard formulas. Bradfield's examples are:

Theorem 7. *The Σ_n^μ formula*

$$\mu X_n.\nu X_{n-1}\ldots\theta X_1.[c]X_1 \vee \langle R_1\rangle X_1 \vee \ldots \vee \langle R_n\rangle X_n,$$

where $\theta = \mu$ if n is odd and $\theta = \nu$ otherwise, is not equivalent to any Π_n^μ formula.

A later, very elegant proof of the strictness of the hierarchy is given by Arnold in [26]. The idea is to consider certain formulas W_n, describing winning positions for the first player of a parity game, and to reduce each formula of class Σ_n^μ to W_n via a contraction in the complete metric space of binary trees. Since all contractions in a complete metric space have a fixpoint, a diagonal argument shows that W_n cannot be of class Π_n^μ.

7.5. Monadic logic

Recall that *monadic second order logic* is first order logic plus quantifiers on sets. The definition of the semantics implies that the modal μ-calculus is included into monadic second order logic: in fact, a point x belongs to $\mu Y.\phi(Y)$ iff x belongs to all sets E such that $E = \phi(E)$, and x belongs to $\nu Y.\phi(Y)$ iff x belongs to some set E such that $E = \phi(E)$. Moreover, the formulas of modal μ-calculus are invariant under bisimulation, as can be seen, for example, by translating modal μ-calculus into infinitary modal logic as we did in a previous section.

There is a nice converse to these remarks, due to Janin and Walukiewicz, that is:

Theorem 8. *On arbitrary graphs, every formula of monadic second order logic invariant under bisimulation is equivalent to a formula of modal μ-calculus.*

One may ask whether there is an analogous theorem for the levels Σ_n of monadic second order logic and the corresponding levels $\Sigma_n^{N\mu}$ of the modal μ-calculus. The answer is given in [27–29]: the correspondence holds for $n = 0, 1, 2$ ($n = 0$ is a classical modal logic result of Van Benthem, see [30]) but fails for $n \geq 3$, because the formula W_n is in Σ_3, but is not in $\Sigma_{n-1}^{N\mu}$.

An interesting open problem is whether the theorem above, or its n^{th} versions, hold over *finite* graphs. So far we know only that the modal case ($n = 0$) is true, see [31].

Another challenging question is whether, over arbitrary graphs, the modal μ-calculus is included in Σ_3. This is known to be true over several subclasses of graphs, for instance over trees and over graphs of finite fixed degree, see [27].

7.6. Automata

The history of automata-theoretic approaches to verification and associated fields has involved ever more invention of new types of automata, as it becomes apparent that some particular restriction is technically inconvenient and can be lifted without harm. In order to explain the main results, we shall need to introduce several of these varieties; let us start at the beginning.

A *deterministic automaton* on *finite words* is the usual deterministic finite state machine, accepting finite words over its input alphabet; a *run* of such an automaton

is a path through the machine matching the input, and it *accepts* if the final state is accepting. A *nondeterministic* automaton on finite words is the usual notion, namely that the transition function may specify more than one successor state for a given state and input letter; a run of such an automaton requires choosing one successor as each nondeterministic choice, and the automaton accepts its input if some run accepts that input.

There are now several orthogonal generalizations of this idea. Firstly, we may allow *infinite* words as well as finite words. In this case, we must specify acceptance conditions which determine when an infinite run is successful. There are many of these; the most important are as follows. A *Büchi* acceptance condition specifies a set G of states, such that we infinitely often meet a G-state on the run (mnemonic: a green light must flash infinitely often). A *Rabin* acceptance condition specifies k sets R_i and k sets G_i, and requires that for some i, we see G_i infinitely often, and don't see R_i infinitely often (mnemonic: k sets of lights; there must be one set where we see green often and don't see red often). In a *Mostowski* or *parity* acceptance condition, each state of the automaton is assigned an integer rank, and the highest rank occurring infinitely often must be even (there is no general agreement on highest/lowest or even/odd in this definition).

In the next dimension, we may feed the automata *trees* rather than words. To start with, assume that the branching degree is fixed at n. The automaton transition function now specifies n successors, and notionally the automaton splits into n copies as it passes to the successors, so a run is a tree matching the input tree. The run is successful if all paths through the run satisfy the acceptance condition, which is specified as above. In the case of a nondeterministic tree automaton, the transition is chosen nondeterministically, but each transition specifies n successors. We can generalize this to automata on trees whose branching degree varies according to the label, or to trees whose branching degree is freely variable (so called amorphous automata).

Finally, we can complicate the success condition as follows: instead of saying just that every path through the run accepts, we can say that the subrun rooted at a vertex v is accepted if some boolean combination, depending on the state, $f(\{T_i\}_{i\in I})$ holds, where T_i is the statement that the subrun rooted at v's i^{th} child is accepted, and that the run is accepted if the subrun starting at the root is accepted, provided the global acceptance condition is met. The global acceptance condition is most easily formulated game-theoretically, but one can think of it in μ-calculus terms as being: there must exist a choice function which for each vertex selects a satifying assignment for $f(\{T_i\}_{i\in I})$, and thus selects some children; then every path through the selected vertexes must meet the Rabin etc. acceptance condition. These are alternating automata. Warning: this alternation is that between $\exists$ and $\forall$, nothing to do with fixpoint alternation!

If the boolean combination is just conjunction, $f(\{T_i\}_{i\in I}) = \bigwedge_{i\in I} T_i$, we get a normal tree automaton; if it is disjunction, we get a tree automaton that accepts if some path accepts, rather than if all paths accept, giving a form of nondeterminism. It is probably not much of a surprise that:

Theorem 9. *Modal μ-calculus formulas are equivalent to alternating amorphous parity automata (on trees).*

The equivalence is simple: a diamond modality corresponds to a vertex whose boolean success combination is disjunction, a box modality to one whose success combination is conjunction; and if one assigns ranks to the fixpoint operators that are consistent with subformula inclusion, with even ranks for maximal fixpoints and odd for minimal, the parity condition asserts that the obviously induced tree pre-model is well-founded.

A rather more surprising fact is:

Theorem 10. *Ordinary Rabin automata are equivalent to alternating parity automata (on trees).*

It is obvious that a Rabin automaton can be turned into an alternating parity automaton. However, the reverse construction is not easy. It is quite easy to turn a parity condition into a Rabin condition, but it is harder to remove the alternation, and a large size blowup is required. In terms of modal μ-calculus formulas this is precisely the transformation into automaton normal form, anf, mentioned earlier, which has no $\forall/\exists$ alternation. As a corollary, we have:

Theorem 11. *Rabin automata are equivalent to modal μ-calculus formulas (on trees).*

The theory of automata on infinite objects is highly developed. Wolfgang Thomas [32] provides a survey of the entire area, and Niwiński's [33] is a fundamental study of automata and fixpoint logics, including much useful background material. Here we shall just mention a few more of the immediate connections with the modal μ-calculus:

- In Rabin's original paper [34], one of the hardest lemmas was proving that Rabin automata are closed under complementation. Given the above, it is now obvious, since the modal μ-calculus is closed under complementation by definition.
- Rabin automata have the nice property that if they accept some tree, they accept some regular tree: that is, the tree unraveling of a finite system. This gives the finite model property of the modal μ-calculus, as we have seen.
- The emptiness problem for Rabin automata is decidable. Hence one can model-check by forming the product of the system with the Rabin automaton, and checking for emptiness, as we have also seen.

We have mentioned mostly Rabin automata. Büchi automata on *trees* are strictly less expressive than Rabin automata. However, on *words*, parity, Rabin and Büchi automata are equivalent, and one has:

Theorem 12. *Rabin/Büchi/parity automata on infinite words are equivalent to modal μ-calculus.*

7.7. Parity games

Games have played an important role in many areas of mathematics, but especially in logic. The so-called *Ehrenfeucht-Fraïssé games* characterize first-order logic, and extensions and restrictions of them characterize various extensions and

restrictions of first-order logic. In a similar manner, we can define games for the modal μ-calculus. Technically, such games are fairly trivial reformulations of automata or tableaux; but they have good explanatory power, and, more surprisingly, provide useful approaches to model checking. Owing to the extensive use of games in finite model theory, game formulations of modal μ-calculus problems assist in exploring the close relationships with finite model theory.

A *parity game* is played by two players, whom we call I and II. An *arena* for the game is a graph, in which each vertex is labeled I or II, and also labelled with an integer rank from 1 to k, where k is called the index of the game. A *play* of the game consists of a sequence of moves in which the appropriate player chooses a successor of the current vertex; II wins if the play ends in I getting stuck, or if the play is infinite and the highest rank occurring infinitely often is even. It may be convenient to designate certain vertexes as immediate wins of I or II, for example to deal with atoms in the modal μ-calculus.

A *strategy* for one player is a function which given a partial play from which the player is due to move, gives the next move. A *winning strategy* is one which if followed guarantees a win. A strategy is *history-free* (or *memoryless*) if it depends only on the current position in the game. In parity games, it is always the case that one player has a winning strategy, and even a memoryless winning strategy; so, on finite graphs, the winner can be explicitly computed.

It turns out that modal μ-calculus model checking can be viewed as a parity game in a natural way, so that a structure satisfies a formula iff player I wins the associated game. This gives a reduction from model checking to parity games. A converse reduction can be obtained by using the formulas W_n mentioned earlier.

So, it turns out that the two problems are tightly related: they are linear time, many-one reducible to each other. In particular, if one of them is in P, then both are in P.

8. Model checking

What turned out to be one of the more useful techniques for automated reasoning about reactive systems began with the advent of efficient *model checking*, see [10] (*cf.* [35–37]). The basic idea is that the global state transition graph of a finite state reactive system defines a Kripke structure in the sense of modal or temporal logic (*cf.* [9]), and we can give an efficient algorithm for checking if the state graph defines a model of a given specification expressed in an appropriate modal or temporal logic, which most often is contained in the modal μ-calculus. While earlier work in the protocol community had addressed the problem of analysis of simple reachability properties, model checking provided an expressive, uniform specification language in the form of modal or temporal logic along with a single, efficient verification algorithm which automatically handled a wide variety of correctness properties.

8.1. Taxonomy of model checking approaches

It is possible to give a rough taxonomy of model checking methods according to certain criteria:

- *Explicit state representation versus symbolic state representation.* In the explicit state approach the Kripke structure is represented extensionally using conventional data structures such that adjacency matrices and linked lists, so that each state and transition is enumerated explicitly. In contrast, in the symbolic approach, boolean expressions denote large Kripke structures implicitly. Typically, the data structure involved is that of *Binary Decision Diagrams* (BDDs), which can, in many applications, although not always, manipulate boolean expressions denoting large sets of states efficiently.

  The distinction between explicit state and symbolic representations is to a large extent an implementation issue, rather than a conceptual one. The original model checking method was based on an algorithm for fixpoint computation and it was implemented using explicit state representation. The subsequent symbolic model checking method uses the same fixpoint computation algorithm, but now represents sets of states implicitly. However, the succintness of BDD data structures underlying the implementation can make a significant practical difference.

- *Global calculation versus local search.* In the global approach, we are given a structure $\mathcal{K}$ and a formula ϕ. The algorithm calculates $\|\phi\|^{\mathcal{K}}$, the set of all vertexes where ϕ is true. This necessarily entails examining the entire structure. Global algorithms typically proceed by induction on the formula structure, calculating $\|\psi\|^{\mathcal{K}}$ for the various subformulas ψ of ϕ. The algorithm can be presented in recursive form; as the recursion unwinds, the values of the shortest formula are calculated first, then the next shortest, etc.

  In contrast, in the local approach, we are given a specific vertex v_0 in $\mathcal{K}$ along with ϕ. We wish to determine whether $v_0 \in \|\phi\|^{\mathcal{K}}$. The computation proceeds by performing a search of $\mathcal{K}$ starting at v_0. The potential advantage is that, many times in practice, only a portion of $\mathcal{K}$ may need to be examined to settle the question. In the worst case, however, it may still be necessary to examine all of $\mathcal{K}$ (see [38]).

- *Monolithic structures versus incremental algorithms.* To some extent this is also more of an implementation issue than a conceptual one. Again, however, it can have significant practical consequences. In the monolithic approach, the entire structure $\mathcal{K}$ is built and represented at one time in computer memory. While conceptually simple and consistent with standard conventions for judging the complexity of graph algorithms, in practice this may be highly undesirable because the entire graph of $\mathcal{K}$ may not fit in computer memory at once. In contrast, the incremental approach (also referred to as the on-the-fly or on-line approach) entails building and storing only small portions of the graph of $\mathcal{K}$ at any one time (see [39]).

8.2. Complexity of explicit state model checking

There is a naive, recursive algorithm for evaluation of a formula ϕ of the modal μ-calculus on a finite structure $\mathcal{K}$. The idea is that, to compute the semantics of a fixpoint $\mu X.f(X)$ or $\nu X.f(X)$, we initialize the set variable X to false or true respectively, and while X is different from the semantics of $f(X)$, we replace X with

the semantics of $f(X)$. By Theorem 1, the while loop terminates in at most $|\mathcal{K}|$ steps, and upon termination, the value of X coincides with the semantics of the fixpoint. The naive algorithm can be so implemented to have time complexity $(|\mathcal{K}|\,|\phi|)^{O(ad(\phi))}$, so it is exponential in general, but polynomial if restricted to formulas with fixed alternation depth. Since all practical correctness properties seem to be of alternation depth 1 or 2, we have a low order polynomial time algorithm. Clever variants of the naive algorithm exist, which achieve polynomial *space* complexity, namely $O(|\mathcal{K}|\,|\phi|)$, but they still have time complexity no better than $(|\mathcal{K}|\,|\phi|)^{O(ad(\phi))}$.

What is the problem with this complexity? Lichtenstein and Pnueli [40] advance the following argument: in practice, it is typically the *structure* size, rather than the formula size, that is the dominant factor in the complexity, because structures are usually extremely large, while specifications are often rather short. Hence, it is highly desirable to have an algorithm whose complexity grows linearly in the structure size, while even exponential growth in the specification size may be tolerable.

In particular, for alternation depth 1, we can get algorithms of time $O(|\mathcal{K}|\,|\phi|)$. So, a natural question is:

Problem 1. *Is there a model checking algorithm for alternation depth* 2 *which runs in time linear in the structure size?*

In terms of complexity classes, by [41] we have that the model checking for the modal μ-calculus is in NP (thus also in $co-NP$ by complementation.) A recent, better upper bound is UP, see [42], where UP is the set of decision problems solvable in polynomial time on a non-deterministic Turing machine where there exists a *unique* accepting path if the string is accepted (thus, UP lies between P and NP). The proof uses the reduction of the problem of model checking to the problem of determining the winner in parity games. But the problem remains:

Problem 2. *Is there a polynomial model checking algorithm for the entire modal μ-calculus?*

8.3. State explosion

We emphasize that the above discussion focuses on extensional model checking, where it is assumed that the structure $\mathcal{K}$ including all of its vertexes and edges are explicitly represented, using data structures such as adjacency lists or adjacency matrices. An obvious limitation then is the combinatorial *state explosion* problem. Given a reactive system composed on n sequential processes running in parallel, its global state graph will be essentially the product of the individual local process state graphs. The number of global states thus grows exponentially in n. For particular systems it may happen that the final global state graph is of a tractable size, say a few hundred thousand states plus transitions. A number of practical systems can be modeled at a useful level of abstraction by state graphs of this size, and extensional model checking can be a helpful tool.

On the other hand, it can quickly become infeasible to represent the global state graph for large n. Even a banking network with 100 automatic teller machines each having just 10 local states, could yield a global state graph of astronomical size amounting to about 10^{100} states.

Plainly, for such astronomical size systems it is out of the question to perform model checking over them, even using algorithms that run in time and space linear in the size of the state space. Various approaches to ameliorating state explosion are currently under investigations. One approach is to use *abstraction*. The basic idea is to replace a large, detailed system $\mathcal{K}$ by a small, less detailed system $\mathcal{K}'$ where inessential information has been suppressed. If an appropriate correspondence between the large and small systems can be established, then correctness of the small system may be used to ensure correctness of the large system.

For instance, suppose there is a homomorphism $h : \mathcal{K} \longrightarrow \mathcal{K}'$ such that v and $h(v)$ agree on the atomic propositions of a linear time formula ϕ, and such that if (v,w) is a transition in $\mathcal{K}$ then $(h(v),h(w))$ is a transition in $\mathcal{K}'$. We may then conclude that if there is a path satisfying $\neg\phi$ in $\mathcal{K}$, then there is an image path satisfying $\neg\phi$ in $\mathcal{K}'$. Hence, if in the small system $\mathcal{K}'$ we have $h(v_0) \models A\phi$, then in the large system we have $\mathcal{K}, v_0 \models A\phi$ as well.

Another approach is to represent transition relations and sets of states symbolically, as described below.

8.4. Symbolic approaches

A noteworthy advance has been the introduction of *symbolic model checking* techniques (see [43–46]) which are – in practice – often able to succintly represent and model check over state graphs of size 10^{100} states and even considerably larger. The basic algorithms used for symbolic model checking are the same as those used for extensional model checking, and are based on iterative calculation of (a representation of) the set of states where each basic modality holds, using fixpoint computation justified by the Theorem 1. The key distinction is that the state graph of the Kripke structure and sets of states where formulas are true in it are represented in terms of a boolean characteristic function which is in turn represented by an (ordered) Binary Decision Diagram (BDD) (*cf.* [47]). These BDDs can in practice be extremely succinct. BDD-based model checkers have been remarkably effective and useful for debugging and verification of hardware circuits. For reasons not well understood, BDDs are often able to exploit the regularity that is readily apparent even to the human eye in many hardware designs. Because software typically lacks this regularity, BDD-based model checking seems much less helpful for software verification. We refer the reader to [43] for an extended account of the utility of BDDs in hardware verification.

It should be emphasized, however, that BDD based model checking methods, are, in worst case, still intractably inefficient. On the one hand, for some structures $\mathcal{K}$ of astronomical size there are small BDDs representing them, and this is exploited in applications as noted above. But for other structures $\mathcal{K}$, sometimes those derived from applications such as software, the BDD representation is intractably large. Plainly, a counting argument shows that most structures do not have a small BDD representation. In any event, checking simple graph reachability in a structure $\mathcal{K}$, *i.e.* $\mathcal{K}, v_0 \models EFQ$, where Q is an atom and $\mathcal{K}$ is represented by a BDD, is PSPACE-complete (*cf.* [47, 48]). The disparity between theoretical, worst case results for symbolic model checking and its surprisingly good performance in practice, has so far militated against the development of an associated complexity theory for this application.

8.5. Debugging versus verification

Model checkers are a type of decision procedure and provide yes/no answers. It turns out that, in practice, model checkers are often used for debugging as well as verification. In industrial environments it seems that the capacity of a model checker to function as a debugger is perhaps better appreciated than their utility as a tool for verifying correctness.

Consider the empirical fact that most designs are initially wrong and must go through a sequence of corrections/refinements before a truly correct design is finally achieved. Suppose one aspect of correctness that we wish to check is that if a system is started in a fixed "good" vertex v_0, then it remains forever in "good" vertexes; that is, in *CTL* terms, v_0 verifies $AGgood$. It seems quite likely that this invariance may in fact not hold of the initial faulty design, due to conceptually minor but tricky errors in the fine details. Thus, during many iterations of the design process, we have in fact that a "non-good" vertex is reachable from v_0, that is, $\mathcal{K}, v_0 \models EF\neg good$.

It would be desirable to circumvent the global strategy of examining all of $\mathcal{K}$ to calculate the set $\|EF\neg good\|^{\mathcal{K}}$ and then checking whether v_0 is a member of that set. If there does exist a "non-good" vertex reachable from v_0, once it is detected it is no longer necessary to continue the search examining $\mathcal{K}$. This is the heuristic motivating local model checking algorithms. Many of them involve searching from the start vertex v_0 looking for confirming or refuting vertexes or cycles; once found, the algorithm can terminate often prematurely having determined that the formula must be true or must be false at v_0 on the basis of the portion of $\mathcal{K}$ examined during the limited search.

Of course, it may be that all vertexes must be examined before finding a refutation to $AGgood$. Certainly, once a truly correct design is achieved, all vertexes reachable from v_0 must be examined. But in many practical cases, a refutation may be found quickly after limited search.

We note in passing that some symbolic model checkers have been adapted to provide some sort of counter example facility for debugging.

References

[1] Kozen D 1983 *Theoret. Comput. Sci.* **27** 333
[2] Arnold A and Niwiński D 2001 *Rudiments of μ-calculus*, North-Holland
[3] Janin D and Walukiewicz I 1996 *Proc. CONCUR '96, Lecture Notes in Comput. Sci.* **1119** 263
[4] Emerson A and Jutla C 1988 *Proc. of IEEE FOCS*, Piscataway, USA, pp. 328–337
[5] Emerson A 1997 *Proc. of the DIMACS Symp. on Descriptive Complexity and Finite Models*, AMS Press, pp. 185–214
[6] Lenzi G 1997 *The μ-Calculus and the Hierarchy Problem*, PhD Thesis, Scuola Normale Superiore, Pisa
[7] Bradfield J and Stirling C 2001 *Modal Logics and μ-Calculi: an Introduction*, in: *Handbook of Process Algebra*, Elsevier, Chap. 4, pp. 293–330
[8] Dam M 1994 *Theoret. Comput. Sci.* **126** 77
[9] Pnueli A 1977 *Proc. 18th IEEE FOCS*, Providence, USA, pp. 46–57
[10] Clarke E and Emerson A 1982 *Proc. of the Workshop on Logics of Programs, Lecture Notes in Comput. Sci.* **131** 52
[11] Emerson A and Halpern J 1986 *J. of the ACM* **33** (1) 151
[12] Pratt V 1976 *Proc. of FOCS '76*, Houston, USA, pp. 109–121
[13] Streett R and Emerson A 1989 *Inform. and Comput.* **81** 249

[14] Stirling C and Walker D 1991 *Theoret. Comput. Sci.* **89** 161
[15] Bradfield J 1991 *Verifying Temporal Properties of Systems*, Birkhäuser
[16] Bradfield J and Stirling C 1992 *Theoret. Comput. Sci.* **96** 157
[17] Walukiewicz I 1995 *Proc. 10th IEEE LICS*, San Diego, USA, pp. 14–24
[18] Janin D and Walukiewicz I 1995 *Proc. MFCS '95, Lecture Notes in Comput. Sci.* **969** 552
[19] Walukiewicz I 1995 *BRICS Notes Series 95-1*, http://www.brics.dk/BRICS/NS/95/1/BRICS-NS-95-1/index.html
[20] Emerson A and Lei C 1986 *Proc. 1st IEEE LICS*, Cambridge, USA, pp. 267–278
[21] Niwiński D 1986 *Proc. ICALP '86, Lecture Notes in Comput. Sci.* **226** 464
[22] Arnold A and Niwiński D 1990 *J. Inform. Process. Cybernet.* EIK **26** 451
[23] Bradfield J 1997 *Theoret. Comput. Sci.* **195** 133
[24] Lenzi G 1996 *Proc. ICALP '96, Lecture Notes in Comput. Sci.* **1099** 87
[25] Bradfield J 1998 *Proc. STACS '98, Lecture Notes in Comp. Sci.* **1373** 39
[26] Arnold A 1999 *RAIRO-Theoretical Informatics and Applications* **33** 329
[27] Janin D and Lenzi G 2001 *Proc. of LICS*, Boston, USA, pp. 347–356
[28] Janin D and Lenzi G 2002 *Computing and Informatics* **21** 185
[29] Janin D and Lenzi G 2004 *Fund. Inform.* **61** (3-4) 247
[30] van Benthem J 1976 *Modal Correspondence Theory*, PhD Thesis, University of Amsterdam
[31] Rosen E 1997 *J. of Logic, Language and Computation* **6** (4) 427
[32] Thomas W 1998 *Languages, Automata and Logic. Handbook of Formal Language Theory*, vol. III, Springer-Verlag, New York, pp. 389-455
[33] Niwiński D 1997 *Theoret. Comput. Sci.* **189** 1
[34] Rabin M 1969 *Trans. Amer. Math. Soc.* **141** 1
[35] Emerson A 1981 *Branching Time Temporal Logics and the Design of Correct Concurrent Programs*, PhD Thesis, Division of Applied Sciences, Harvard University
[36] Queille J and Sifakis J 1982 *Proc. 5th Int. Symp. Prog., Lecture Notes in Comput. Sci.* **137** 195
[37] Clarke E, Emerson A and Sistla P 1983 *10th ACM Symp. on Principles of Prog. Lang.*; journal version: 1986 *ACM Trans. on Prog. Lang. and Sys.* **8** (2) 244
[38] Stirling C and Walker D 1989 *Lecture Notes in Comput. Sci.* **351** 369
[39] Jard C and Jeron T 1989 *Int. Workshop on Automatic Verification Methods for Finite State Systems, Lecture Notes in Comput. Sci.* **407** 189
[40] Lichtenstein O and Pnueli A 1985 *Proc. of POPL '85*, New Orleans, USA, pp. 97–107
[41] Emerson A, Jutla C and Sistla P 1993 *Proc. CAV '93, Lecture Notes in Comput. Sci.* **697** 385
[42] Jurdzinski M 1998 *Inf. Process. Lett.* **68** 119
[43] McMillan K 1992 *Symbolic Model Checking: an Approach to the State Explosion Problem*, PhD Thesis, Carnegie-Mellon University
[44] Burch J, Clarke E, McMillan M, Dill D and Hwang L 1990 *Proc. of the 5th Annual IEEE-CS Symposium on Logic in Computer Science*, Philadelphia, USA, pp. 428–439
[45] Pixley C 1990 *CAV '90 DIMACS series* **3** (also DIMACS Tech. Report 90-31)
[46] Coudert O and Madre J 1990 *Computer Aided Verification '90*, DIMACS Series, pp. 75–84
[47] Bryant R 1986 *IEEE Trans. on Computers* **C-35** (8) 677
[48] Galperin H and Wigderson A 1983 *Information and Control* **56** 183