Modal Logics for Processes – WORKING DOCUMENT

Julian Bradfield and Colin Stirling

Contents

1 Introduction

In the chapter, we shall first briefly survey the background, history and some of the issues arising from the use of modal logics for processes. We briefly introduce earlier modal logics such as Hennessy-Milner logic, PDL and CTL. We then devote the major part of the chapter to the modal mu-calculus. It is a logic which, although much studied and also widely used, has a reputation for being hard to understand compared to, say, CTL. We therefore devote a section to introducing the logic and developing an intuition for it. In section 4 we look at the theoretical basis, and major results, in a little more detail. However, much of the detailed technical work in this area has already been covered in other handbook articles [55, 16], or in easily accessible original papers; we have therefore chosen to give few results in fine detail, but rather to provide outline proofs which, we hope, cover the essentials, and serve as a helpful map to the detailed proofs elsewhere, but yet remain readable for those who have no wish to go into detail.

We follow this with a survey of results specifically about model-checking, including the notions of local model-checking, and the work on model-checking classes of infinite processes.

Section 6 looks, quite briefly, at the relationship of modal mu-calculus to other formalisms, in particular automata and games. Much of this is of primarily mathematical interest; but it is also of practical interest, as automata have been used for model-checking, and recently games have also been found to have uses in tools.

Finally, we take a look at some of the many possible extensions of modal logic which are found outside the traditional interleaving view of process algebra.

1.1 Background

The application of modal and temporal logics to process calculi is part of a line of program verification going back to the 1960s and program schemes and Floyd–Hoare logic. Originally the emphasis was on *proof*: Floyd–Hoare logic allows one to make assertions about programs, and there is a proof system to verify these assertions. This line of work has, of course, continued and flourished, and today there are highly sophisticated theories for proving properties of programs, and equally sophisticated machine support for these theories. However, the use of proof systems has some disadvantages, and one hankers after a more purely algorithmic approach to simple problems. One technique was pioneered by Manna and Pnueli [40], who turned program properties into questions of satisfiability or validity in first order logic, which can then be attacked by means that are not just proof-theoretic; this idea was later applied by them to linear temporal logics.

During the 1970s, the theory of program correctness was extended by investigating more powerful logics, and studying them in a manner more similar to the traditions of mathematical logic. A family of logics which received much attention was that of dynamic logics, which can be seen as extending the ideas of Hoare logic [50]. Dynamic logics are modal logics, where the different modalities correspond to the execution of different programs—the formula $\langle\alpha\rangle\Phi$ is read as 'it is possible for α to execute and result in a state

satisfying Φ'. The programs may be of any type of interest; the variety of dynamic logic most often referred to is a propositional language in which the programs are built from atomic programs by regular expression constructors; henceforth, Propositional Dynamic Logic, PDL, refers to this logic. PDL is interpreted with respect to a Kripke structure model, formalizing the notion of the global state in which programs execute and which they change—each point in the structure corresponds to a possible state, and programs determine a relation between states giving the changes effected by the programs.

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

Meanwhile, the theory of process calculi was being developed in the late 70s, most notably by Milner [42]. An essential component was the use of labelled Kripke structures ('labelled transition systems') as a raw model of concurrent behaviour. An important difference between the use of Kripke structures here and their use in program correctness was that the states are the behaviour expressions themselves which model concurrent systems and the labels on the accessibility relation (the transitions) are simple actions (and not programs). The criterion for behavioural equivalence of process expressions was defined in terms of observational equivalence (and later in terms of bisimulation relations). Hennessy and Milner introduced a primitive modal logic in which the modalities refer to actions: $\langle a\rangle\phi$ 'it is possible to do an a action and then have ϕ be true', and its dual $[a]\phi$ 'ϕ holds after every a action'. Together with the usual boolean connectives, this gives Hennessy–Milner logic [25], HML, which was introduced as an alternative exposition of observational equivalence. However, as a logic HML is obviously inadequate to express many properties, as it has no means of saying 'always in the future' or other temporal connectives—except by allowing infinitary conjunction, which is a trifle drastic.

In 1983, Dexter Kozen published a study of a logic that combined simple modalities, as in HML, with fixpoint operators to provide a form of recursion. This logic, the modal mu-calculus, has become probably the most studied of all temporal logics of programs. It has a simple syntax, an easily given semantics, and yet the fixpoint operators provide immense power. Most other temporal logics, such as the CTL family, can be seen as fragments of the modal mu-calculus. Moreover, this logic lends itself to transparent model-checking algorithms.

Another 'root' to understanding modal logics is the work in the 60s on automata over infinite words and trees by Büchi [**?**] and Rabin [52]. The motivation was decision questions of monadic second-order logics. Büchi introduced automata as a normal form for such formulas. This work founded new connections with logic and automata theory. Later it was realised that modal logics are merely sublogics of appropriate monadic second-order logic, and that the automata normal forms provide a very powerful framework within which to study properties of modal logic. A further development was the use of games by

Gurevich and Harrington [23] as an alternative to automata.

There is also an older game-theoretic tradition due to Ehrenfeucht and Fraïssè, for understanding the expressive power of logics. These techniques are also applicable within process calculi. For instance bisimulation equivalence can be naturally rendered as such a game, and expressivity of modal logics can be understood using game-theroetic techniques.

1.2 The general framework

There is an enormous variety of process calculi, as illustrated in other chapters of this Handbook. For the most part they involve the following; a syntactic description of a process as an expression built from a few combinators, a behavioural meaning, a labelled transition system, determined by structural operational rules, and an equivalence (or better a congruence) to stipulate in what circumstances two expressions count as describing the same system. Using the operational rules a process expression generates a transition graph which may be infinite-state. This graph reflects the potential behaviour that an expression may exhibit.

One systematic approach to capturing 'correctness' in process calculi is through the use of an equivalence; a complex description of a system in terms of all its components may be shown to be equivalent to a small description. However for many complex systems their descriptions are either too large to formally manipulate or there is not a small intended equivalent description. In these cases one may be more interested in ascertaining whether crucial properties hold, such as a variety of liveness and safety properties.

The primary application of modal logics to processes is therefore: given a process expression E, does it have a particular modal property ϕ, that is does $E \models \phi$? More traditional questions about a modal logic may also be interesting. For instance the question 'is ϕ satisfiable?', that is, 'is there a process E such that $E \models \phi$?' could be useful if ϕ captures the requirements of a system. Another traditional question is whether a formula ϕ is valid, that is, does every process E have property ϕ ?

There are many issues involved in the verification question. What kind of expression is E and what is its structure? Which 'control structures' does it employ and what extras does it contain such as time, probability, and locality. What sort of property is ϕ, and which property language does it belong to? Also there are different approaches to the basic problem, does $E \models \phi$?

(1) Direct approach. Develop techniques which can verify directly whether or not $E \models \phi$. Note that there are two kinds of structure, the logical structure of ϕ which may contain boolean operators, modal and temporal operators and fixed points, and the process structure which may contain concurrent composition, synchronization and abstraction mechanisms. Methods for verification may or may not employ 'structured reasoning'.

(2) Indirect approach I. Reduce the verification problem to a validity problem. Code the process E as a formula ψ_E (such as a 'characteristic formula') in the logic. The verification problem thereby reduces to the validity problem, is $\psi_E \rightarrow \phi$ valid?

(3) Indirect approach II. This is similar to (2) but dispenses with logic altogether! A process E is very close to an automaton (or to a game). Consider the characteristic automaton (game) associated with E, call it A_E and consider the automaton (game) for ϕ. This gives a containment problem: is A_E contained in A_ϕ? (Alternatively take the complement automaton of ϕ, A^c_ϕ, and find out if the intersection of A_E and A^c_ϕ is nonempty.)

2 Early logics

HML is a primitive modal logic of action. The syntax of HML has, in addition to the boolean operators, a modality $\langle a \rangle \phi$, where a is a process *action*. A structure for the logic is just a labelled transition system. Atomic formulas of the logic are the constants $\mathtt{tt}$ and $\mathtt{ff}$. The meaning of $\langle a \rangle \phi$ is 'it is possible to do an a-action to a state where ϕ holds'. The formal semantics is given in the obvious way by inductively defining when a state (a process) of a transition system has a property; for example, $E \models \langle a \rangle \phi$ iff $\exists F. E \xrightarrow{a} F \wedge F \models \phi$. As mentioned above the logic was introduced as a means for understanding process equivalence. We may also add some notion of variable or atomic proposition to the logic, in which case we provide a valuation which maps a variable to the set of states at which it holds.

The logic PDL, as mentioned above, is both a development of Floyd–Hoare style logics, and a development of modal logics. It is not widely used these days, but for historical reasons it is worth presenting it briefly. PDL is an extension of HML in the circumstance that the action set has some structure. There is room for variation in the meaning of action, but in the standard logic, a program is considered to have a number of *atomic actions*, which in process algebraic terms are just process actions, and α is allowed to be a regular expression over the atomic actions: a, $\alpha;\beta$, $\alpha \cup \beta$, or α^*. We may consider atomic actions to be uninterpreted atoms; but in the development from Floyd–Hoare logics, one would see the atomic actions as, for example, assignment statements in a **while** program.

PDL enriches the labels in the modalities of HML. An alternative extension of HML is to include further modalities. The branching time logic CTL (Computation Tree Logic) can be described in this way as an extension of HML, with some extra 'temporal' operators which permit expression of liveness and safety properties. For the semantics we need to consider 'runs' of a process. A run of E_0 is a sequence $E_0 \xrightarrow{a_1} E_1 \xrightarrow{a_2} \ldots$ which may have finite or infinite length; if it has finite length then its final process is a 'sink' process which has no transitions. A run $E_0 \xrightarrow{a_1} E_1 \xrightarrow{a_2} \ldots$ has the property $\phi\, \mathbf{U}\, \psi$, 'ϕ until ψ', if there is an $i \geq 0$ such that $E_i \models \psi$ and for all $j : 0 \leq j < i$, $E_j \models \phi$.

$$\begin{array}{ccccccc} E_0 & \xrightarrow{a_1} & E_1 & \xrightarrow{a_2} & \ldots & E_i & \xrightarrow{a_{i+1}} & \ldots \\ \models & & \models & & \ldots & \models & & \\ \phi & & \phi & & & \psi & & \end{array}$$

The formula $\mathbf{F}\phi = (\mathtt{tt}\, \mathbf{U}\, \phi)$ means 'ϕ eventually holds' and $\mathbf{G}\phi = \neg(\mathtt{tt}\, \mathbf{U}\, \neg\phi)$: '$\phi$ always holds'. For each 'temporal' operator such as $\mathbf{U}$ there are two modal variants, a strong variant ranging over all runs of a process and a weak variant ranging over some run of a process. We preface a strong version with $\forall$ and a weak version with $\exists$. If HML is extended with the two kinds of until operator the resulting logic is a slight but inessential variant of CTL. The formal semantics is given by inductively defining when a state (process) has a property. For instance $E \models \forall[\phi \, \mathbf{U} \, \psi]$ iff every run of E has the property $\phi \, \mathbf{U} \, \psi$.

Variants and enrichments: CTL* ECTL*. These extend linear linear time logic. For example, $\forall[P \, \mathbf{U} \, \exists \mathbf{F} Q]$ is a CTL formula, but $\forall[P \, \mathbf{U} \, \mathbf{F} Q]$ is not, because the $\mathbf{F}$ is not immediately governed by a quantifier. These extensions cover traditional temporal logics—that is, literally logics of time—as advocated by Manna and Pnueli and others. In this view, time is a linear sequence of instants, corresponding to the states of just one execution path through the program. One can define a logic on paths which has operators $\bigcirc\phi$ 'in the next instant (on this path) ϕ is true', and $\phi \, \mathbf{U} \, \psi$ 'ϕ holds until ψ holds (on this path)'; and then a system satisfies a formula if all execution paths satisfy the formula. This can be applied to processes.

One can also extend PDL with temporal operators, as in process logic. Add other enrichments: time probability etc..,

2.1 The small model property

A general result about many modal logics is that they have the small model property; that is if a formula is satisfiable, then it is satisfiable by a model of some bounded size. Provided that model-checking is decidable, which is the case for almost all temporal logics (we shall see an exception in the final section), this immediately gives decidability of satisfiability for the logic, as one need simply check every transition system up to the size bound.

The technique used to establish the small model property for PDL (and therefore HML) is a classical technique in modal logic, that of *filtration*. Let E be a possibly infinite-state process with the property ϕ, and let $\mathbf{T}$ be its transition system. Let Γ be the set of all subformulas of ϕ and their negations: in the case of PDL one also counts $\langle\alpha\rangle\psi$ and $\langle\beta\rangle\psi$ as subformulas of $\langle\alpha \cup \beta\rangle\psi$, $\langle\beta\rangle\psi$ as a subformula of $\langle\alpha;\beta\rangle\psi$ and $\langle\alpha;\alpha^*\rangle\psi$, and $\langle\alpha\rangle\psi$ as subformulas of $\langle\alpha^*\rangle\psi$. The size of Γ is proportional to $|\phi|$. One then filters $\mathbf{T}$ through Γ by defining an equivalence relation on the states of $\mathbf{T}$, $E \equiv F$ iff $\forall\Psi \in \Gamma. E \models \psi \Leftrightarrow F \models \psi$. We define the filtered model to have states $\mathcal{T}/\equiv$ and with atomic action relations given by $[E] \xrightarrow{a} [F]$ iff $\exists E' \in [E], F' \in [F]. E' \xrightarrow{a} F'$. The number of equivalence classes is at most $2^{|\Gamma|}$ and so is $O(2^{|\phi|})$. The rest of the proof shows that the filtered model is indeed a model, in that $[E] \models \psi$ iff $E \models \psi$ for $\psi \in \Gamma$. For PDL the only case requiring comment is the case $\langle\alpha^*\rangle\psi$, which proceeds by an induction on the length of the witnessing sequence of α's. Consequently if ϕ is a satisfiable PDL formula, then it has a model with size $O(2^{|\phi|})$, and in fact $2^{|\phi|}$ suffices—see [21] for full details.

In order to obtain an upper bound for satisfiability from the small model property, we also need to know the complexity of model-checking, that is, determining whether $E \models \phi$. It is straightforward to define an inductive procedure for this, which is polynomial in the size of the formula and of the system. For example, to determine the truth of $\langle\alpha^*\rangle\phi$, one computes the *-closure of the α relation, and then checks for an α^*-successor satisfying

ϕ. These results give an NTIME(c^n) upper bound for the satisfiability problem. By a reduction to alternating Turing machines, [21] also gave a lower bound of DTIME($c^{n/\lg n}$). A closer to optimal technique for satisfiability due to Pratt uses tableaux [**?**].

Although CTL, CTL*, ACTL and modal mu-calculus all have the finite model property, the proof of this cannot be shown using filtrations. The first attempts utilised semantic tableaux where one explicitly builds a model for a satisfiable formula which has small size. But such a technique is very particular (although for instance can provide optimality for instance for PDL due [**?**]). A more sophisticated method is needed. AUTOMATA: comments. GAMES?

2.2 CTL model-checking

CTL has some obvious differences from PDL. Firstly, although it is a state-based logic, in the sense that the formulae one writes are defined on states, it uses path operators internally: evaluating the formula $\forall[\phi \, \mathbf{U} \, \psi]$ at a state involves considering all paths from that state. Thus, at first sight, one might expect to lose the obvious exponential upper bound on model-checking. However, this turns out not to happen, and in fact CTL is very easy to model-check. This was shown by a direct construction in [14]; it also follows from the fact that, as we shall see later, CTL is a particularly simple fragment of the modal mu-calculus.

The model-checking procedure of [14] is an example of an apparently global technique. It proceeds by model-checking subformulae from the bottom up, doing a full pass over the state space for subformulae before considering the superformula. In the original paper, the algorithm is presented as imperative pseudo-code, and it takes a little thought to see why it works. Here is an English outline, which extracts the idea of the code:

- to check $\mathtt{tt}, \neg\phi, \phi_1 \wedge \phi_2$, check the subformulae and perform the boolean operation;
- to check $\langle a\rangle\phi, [a]\phi$, check ϕ, and then apply the semantic definitions;
- to check $\exists[\phi \, \mathbf{U} \, \psi]$, check the subformulae, then find the states at which ψ holds, and trace backwards along paths on which ϕ holds; these states satisfy $\exists[\phi \, \mathbf{U} \, \psi]$;
- to check $\forall[\phi \, \mathbf{U} \, \psi]$, check the subformulae; then make a depth-first traversal of the system, applying the following: if a state satisfies ψ, mark it as satisfying $\forall[\phi \, \mathbf{U} \, \psi]$; otherwise, if it fails ϕ, mark it as failing $\forall[\phi \, \mathbf{U} \, \psi]$; otherwise, after processing the successors, mark it as satisfying $\forall[\phi \, \mathbf{U} \, \psi]$ iff all its successors do.

In following sections, we shall see CTL translated into modal mu-calculus, and we shall see global, backward-looking model-checking algorithms, and local, forward-looking algorithms. It is interesting, from today's perspective, to see that this early CTL algorithm has elements of both: the code for $\exists[\phi \, \mathbf{U} \, \psi]$ is doing exactly the computation by approximants (see section 4.4) of the mu-calculus translation; but the code for $\forall[\phi \, \mathbf{U} \, \psi]$ is doing tableau model-checking (see section 5.1).

3 Introduction to mu-calculi

The defining feature of mu-calculi is the use of fixpoint operators.[1] The use of fixpoint operators in program logics goes back at least to De Bakker, Park and Scott [47]. However, their use in modal logics of programs dates from work of Pratt, Emerson and Clarke and Kozen. Pratt's version [51] used a fixpoint operator like the minimization operator of recursion theory, and this has not been further studied. Emerson and Clarke added fixed points to a temporal logic to capture fairness and other correctness properties [17]. Kozen's [32] paper introduced the modal mu-calculus as we use it today, and established a number of basic results[2].

Fixpoint logics are notorious for being incomprehensible. Indeed, it has been known for several reasonably expert people to spend an hour or two trying to work out whether a one line formula means what it was intended to mean. Furthermore, a full understanding of the mathematical intricacies of the modal mu-calculus requires familiarity with mathematics that, although not difficult, is not covered in most undergraduate mathematics or computer science programmes; this can be a source of irritation for the practitioner new to the area. Fortunately, one can get a very good intuitive understanding, which is even provably equivalent to the formal semantics, without knowing the full details. Moreover, this intuition is all but essential for understanding the deep theory as well.

In this section, therefore, we shall give an introduction to the modal mu-calculus that avoids the harder and more technical details, and concentrates rather on developing the intuition. In the first two subsections, we shall be so informal as to use notation and terminology well in advance of their definitions. In the following section, the semantics of the logic is examined in a little more detail, and the intuitions given a precise mathematical meaning.

3.1 Fixpoints as recursion

Suppose that $\mathcal{S}$ is the state space of some system. For example $\mathcal{S}$ could be the set of all processes reachable from by arbitrary length sequences of transitions from some initial process E. One way to provide semantics of a state-based modal logic is to map formulae ϕ to sets of states, that is to elements of $\wp\mathcal{S}$. For any formula ϕ this mapping is given by $\|\phi\|$[3]. The idea is that this mapping tells us at which states each formula holds. If we allow our logic to contain variables with interpretations ranging over $\wp\mathcal{S}$, then we can view the semantics of a formula with a free variable, $\phi(Z)$, as a function $f\colon \wp\mathcal{S} \to \wp\mathcal{S}$. If we take the usual lattice structure on $\wp\mathcal{S}$, given by set inclusion, and if f is a monotonic function,

[1] Whether one should say 'fixpoint', 'fix-point', or 'fixed point' is one of the most controversial questions in the theory of mu-calculi. One of us (Bradfield) takes the view that one has 'fixpoint operators', which generate 'fixed points' of functions. However, few people enunciate with sufficient clarity to distinguish the two in speech, so one can generally avoid pub brawls over the question.

[2] Kozen called the logic 'propositional mu-calculus' but this would be more appropriate for propositional logic + fixpoint operators, a logic we shall comment on later.

[3] The mapping can be either given directly (inductively) or indirectly as the set $\{s \in \mathcal{S} : s \models \Phi\}$.

then by the Knaster–Tarski theorem we know that f has fixed points, and indeed has a unique maximal and a unique minimal fixed point. So we could extend our basic logic with a minimal fixpoint operator μ, so that $\mu Z.\phi(Z)$ is a formula whose semantics is the least fixed point of f; and similarly a maximal fixpoint operator ν, so that $\nu Z.\phi(Z)$ is a formula whose semantics is the greatest fixed point of f.

The reason we might want to do this is that, as in domain theory for example, it provides a semantics for *recursion*. Writing recursive modal logic formulae may not be an immediately obvious thing to do, but it provides a neat way of expressing all the usual operators of temporal logics. For example, consider the CTL formula $\forall\mathbf{G}\phi$, 'always ϕ'. Another way of expressing this is to say that it is a property X such that if X is true, then ϕ is true, and wherever we go next, X remains true; so X satisfies the modal equation

$$X = \phi \wedge [-]X.$$

So we could write this formula as $?X.\phi \wedge [-]X$. But what is '?'? Which fixed point is the right solution of the equation—least, greatest, or something in between? We may argue thus: if a state satisfies *any* solution X' of the equation, then surely it satisfies $\forall\mathbf{G}\phi$. Hence the meaning of the formula is the largest solution, $\nu X.\phi \wedge [-]Z$.

On the other hand, consider the CTL property $\exists\mathbf{F}\phi$, 'there exists a path on which ϕ eventually holds'. We could write this recursively as 'Y holds if either ϕ holds now, or there's some successor on which Y is true':

$$Y = \phi \vee \langle - \rangle Y.$$

This time, it is perhaps less obvious which solution is correct. However, we can argue that if a state satisfies $\exists\mathbf{F}\phi$, then it surely satisfies any solution Y' of the equation; and so we want the least such solution.

The astute reader will by now have noticed that the equations do not capture the meaning of the English recursive definitions—we should have written

$$X \Rightarrow \phi \wedge [-]X$$

and

$$Y \Leftarrow \phi \vee \langle - \rangle Y,$$

or in terms of the semantic function, $\|X\| \subseteq \|\phi \wedge [-]X\|$ and $\|Y\| \supseteq \|\phi \vee \langle - \rangle Y\|$. So we are not really interested in fixed points, but rather in respectively *post-fixed points* and *pre-fixed points* of the functions defined by the formulae on the right of the (in)equations; and since there is only one canonical post-fixed point and one canonical pre-fixed point, it becomes obvious which is required. It is an easy fact that the greatest post-fixed point and the least pre-fixed point of a monotonic function are also the greatest and least fixed points, and so we may continue to use the notation μ and ν, and work with fixpoints. However, it is usual to emphasize this point when writing the semantics of fixpoint logics by writing, for example,

$$\|\mu Z.\phi(Z)\| = \bigcap\{\, S \mid S \supseteq \|\phi(S)\| \,\}$$

with inclusion rather than equality.

At this point, the reader who does not already know the answer might like to apply a similar argument to produce the fixpoint translation of the CTL formula $\forall[\phi\,\mathbf{U}\,\psi]$.

3.2 Approximating fixpoints and μ as 'finitely'

The other key idea is that of approximants and unfolding. The standard theory tells us that if f is a monotonic function on a lattice, we can construct[4] the least fixed point of f by iterating f on the bottom element $\bot$ of the lattice to form an increasing chain whose limit is the fixed point. The length of the iteration is in general transfinite, but is bounded at worst by the cardinal after cardinality of the lattice, and in the special case of a powerset lattice $\wp\mathcal{S}$, by the cardinal after the cardinality of $\mathcal{S}$. So if f is monotonic on $\wp\mathcal{S}$, we have

$$\mu f = \bigcup_{\alpha<\kappa} f^\alpha(\emptyset)$$

and similarly

$$\nu f = \bigcap_{\alpha<\kappa} f^\alpha(\mathcal{S})$$

—or in terms of a infinitary logic, $\mu Z.\phi(Z) = \bigvee_{\alpha<\kappa} \phi^\alpha(\mathrm{ff})$—where κ is at worst $|\mathcal{S}|+1$ for finite $\mathcal{S}$, or ω_1 for countable $\mathcal{S}$. So for a minimal fixpoint $\mu Z.\phi(Z)$, if a state s satisfies the fixpoint, it satisfies some approximant, say for convenience the $\beta{+}1$ th so that $s \models \phi^{\beta+1}(\mathrm{ff})$. Now if we *unfold* this formula once, we get $s \models \phi(\phi^\beta(\mathrm{ff}))$. That is, the fact that s satisfies the fixpoint depends, via ϕ, on the fact that other states satisfy the fixpoint *at smaller approximants than s does.* These states in turn depend on states that satisfy the fixpoint at still smaller approximants, bringing to mind the well-known doggerel

> Big fleas have little fleas
>   upon their backs to bite 'em
> and little fleas have smaller fleas
>   and so *ad infinitum*

However, in this case the doggerel is wrong: the process cannot continue *ad infinitum*, because the ordinals are well-founded and there is a smallest flea, namely the zeroth approximant $\phi^0(\mathrm{ff})$. So if one follows a chain of dependencies, the chain terminates. This is the strict meaning behind the slogan "μ means 'finite looping'", which, with a little refinement, is sufficient to understand the mu-calculus.

On the other hand, for a maximal fixpoint $\nu Z.\phi(Z)$, there is no such decreasing chain: $s \models \nu Z.\phi(Z)$ iff $s \models \phi(\nu Z.\phi(Z))$, and we may loop for ever, as in the CCS process $P \stackrel{\text{def}}{=} a.P$, which satisfies $\nu Z.\langle a\rangle Z$. (However, if a state *fails* a maximal fixpoint, then there is a descending chain of failures.) Instead, we have the somewhat mystical principle of fixpoint induction: if by assuming that a set $S \models Z$, we can show that $S \models \phi(Z)$, then

[4] We remark that this chapter is entirely set in classical *ZFC*, and no 'construction' is effective unless explicitly stated.

we have shown that $S \models \nu Z.\phi$ (compare the recursive formulation of $\forall\mathbf{G}\phi$ in the previous section).

So in summary, one may understand fixpoints by the slogan 'ν means looping, and μ means finite looping'. This slogan provides an alternative means of explaining why a minimal fixpoint is required in the translation of $\exists\mathbf{F}\phi$. This formula means that there is a path on which ϕ eventually holds: that is, on the chosen path, ϕ holds within finite time. Hence the 'equation' $Y = \phi \vee \langle -\rangle Y$ must only be applied a finite number of times, and so by the slogan we should use a minimal fixpoint.

In the case of formulae with alternating fixpoints (which we shall examine a little later), the slogan remains valid, but requires a little more care in application. It is essential to almost all proofs about modal mu-calculus: the notion of 'well-founded premodel' with which Streett and Emerson [60] proved the finite model property, is an example of the slogan; so are the tableau model-checking approaches of Stirling and Walker, and Bradfield and Stirling.

3.3 Syntax of modal mu-calculus

Let Var be an (infinite) set of *variable names*, typically indicated by $Z, Y, \ldots$; let Prop be a set of *atomic propositions*, typically indicated by $P, Q, \ldots$; and let $\mathcal{L}$ be a set of *labels*, typically indicated by $a, b, \ldots$. The set of modal mu-calculus formulae (with respect to $\mathrm{Var}, \mathrm{Prop}, \mathcal{L}$) is defined in parsimonious form as follows:

- P is a formula.
- Z is a formula.
- If ϕ_1 and ϕ_2 are formulae, so is $\phi_1 \wedge \phi_2$.
- If ϕ is a formula, so is $[a]\phi$.
- If ϕ is a formula, so is $\neg\phi$.
- If ϕ is a formula, then $\nu Z.\phi$ is a formula, provided that every free occurrence of Z in ϕ occurs positively, i.e. within the scope of an even number of negations. (The notions of free and bound variables are as usual, where ν is the only binding operator.)

If a formula is written as $\phi(Z)$, it is to be understood that the subsequent writing of $\phi(\psi)$ means ϕ with ψ substituted for all free occurrences of Z. There is no suggestion that Z is the only free variable of ϕ; and variable capture is not avoided.

The positivity requirement on the fixpoint operator is a syntactic means of ensuring that $\phi(Z)$ denotes a functional monotonic in Z, and so has unique minimal and maximal fixpoint. This requires irritating book-keeping, and it is usually more convenient to introduce derived operators defined by de Morgan duality, and work in positive form:

- $\phi_1 \vee \phi_2$ means $\neg(\neg\phi_1 \wedge \neg\phi_2)$.

- $\langle a\rangle\phi$ means $\neg[a]\neg\phi$.

- $\mu Z.\phi(Z)$ means $\neg\nu Z.\neg\phi(\neg Z)$.

Note the triple use of negation in μ, which is required to maintain the positivity. A formula is said to be in *positive form* if it is written with the derived operators so that $\neg$ only occurs applied to atomic propositions. It is in *positive normal form* if in addition all bound variables are distinct. Any formula can be put into positive normal form by use of de Morgan laws and α-conversion. So we shall often assume positive normal form, and when doing structural induction on formulae will often take the derived operators as primitives.

For the concrete syntax, we shall assume that modal operators have higher precedence than boolean, and that fixpoint operators have lowest precedence, so that the scope of a fixpoint extends as far to the right as possible.

There are a few extensions to the syntax which are convenient in presenting examples, and in practice. The most useful is to allow modalities to refer not just to single actions, but to sets of actions. The most useful set is 'all actions except a'. So:

- $s \models [K]\phi$ iff $\forall a \in K. s \models [a]\phi$, and $[a, b, \ldots]\phi$ is short for $[\{a, b, \ldots\}]\phi$.

- $[-K]\phi$ means $[\mathcal{L} - K]\phi$, and set braces may be omitted.

Thus $[-]\phi$ means just $[\mathcal{L}]\phi$.

3.4 Semantics of modal mu-calculus

A modal mu-calculus *structure* $\mathcal{T}$ (over $\mathrm{Prop}, \mathcal{L}$) is a labelled transition system, namely a set $\mathcal{S}$ of states and a transition relation $\rightarrow \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$ (as usual we write $s \xrightarrow{a} t$), together with an interpretation $\mathcal{V}_{\mathrm{Prop}}\colon \mathrm{Prop} \rightarrow \wp\mathcal{S}$ for the atomic propositions.

Given a structure $\mathcal{T}$ and an interpretation $\mathcal{V}\colon \mathrm{Var} \rightarrow \wp\mathcal{S}$ of the variables, the set $\|\phi\|^{\mathcal{T}}_{\mathcal{V}}$ of states satisfying a formula ϕ is defined as follows:

$$
\begin{array}{rcl}
\|P\|^{\mathcal{T}}_{\mathcal{V}} & = & \mathcal{V}_{\mathrm{Prop}}(P) \\
\|Z\|^{\mathcal{T}}_{\mathcal{V}} & = & \mathcal{V}(Z) \\
\|\neg\phi\|^{\mathcal{T}}_{\mathcal{V}} & = & \mathcal{S} - \|\phi\|^{\mathcal{T}}_{\mathcal{V}} \\
\|\phi_1 \wedge \phi_2\|^{\mathcal{T}}_{\mathcal{V}} & = & \|\phi_1\|^{\mathcal{T}}_{\mathcal{V}} \cap \|\phi_2\|^{\mathcal{T}}_{\mathcal{V}} \\
\|[a]\phi\|^{\mathcal{T}}_{\mathcal{V}} & = & \{\, s \mid \forall t. s \xrightarrow{a} t \Rightarrow t \in \|\phi\|^{\mathcal{T}}_{\mathcal{V}} \,\} \\
\|\nu Z.\phi\|^{\mathcal{T}}_{\mathcal{V}} & = & \bigcup\{ S \subseteq \mathcal{S} \mid S \subseteq \|\phi\|^{\mathcal{T}}_{\mathcal{V}[Z:=S]}\}
\end{array}
$$

where $\mathcal{V}[Z := S]$ is the valuation which maps Z to S and otherwise agrees with $\mathcal{V}$. If we are working in positive normal form, we may add definitions for the derived operators by duality:

$$
\begin{array}{rcl}
\|\phi_1 \vee \phi_2\|^{\mathcal{T}}_{\mathcal{V}} & = & \|\phi_1\|^{\mathcal{T}}_{\mathcal{V}} \cup \|\phi_2\|^{\mathcal{T}}_{\mathcal{V}} \\
\|\langle a\rangle\phi\|^{\mathcal{T}}_{\mathcal{V}} & = & \{\, s \mid \exists t. s \xrightarrow{a} t \wedge t \in \|\phi\|^{\mathcal{T}}_{\mathcal{V}} \,\} \\
\|\mu Z.\phi\|^{\mathcal{T}}_{\mathcal{V}} & = & \bigcap\{ S \subseteq \mathcal{S} \mid S \supseteq \|\phi\|^{\mathcal{T}}_{\mathcal{V}[Z:=S]}\}
\end{array}
$$

We drop $^{\mathcal{T}}$ and $_{\mathcal{V}}$ whenever possible; and write $s \models \phi$ for $s \in \|\phi\|$.

We have discussed informally the importance of approximants; let us now define them. If $\mu Z.\phi(Z)$ is a formula, then for $\alpha \in \mathrm{On}$, let $\mu Z^{\alpha}.\phi$ and $\mu Z^{<\alpha}.\phi$ be formulae, with semantics given, with simultaneous induction on α, by:

$$
\begin{array}{rcl}
\|\mu Z^{<\alpha}.\phi\|^{\mathcal{T}}_{\mathcal{V}} & = & \bigcup_{\beta<\alpha} \|\mu Z^{\beta}.\phi\|^{\mathcal{T}}_{\mathcal{V}} \\
\|\mu Z^{\alpha}.\phi\|^{\mathcal{T}}_{\mathcal{V}} & = & \|\phi\|^{\mathcal{T}}_{\mathcal{V}[Z:=\|\mu Z^{<\alpha}.\phi\|^{\mathcal{T}}_{\mathcal{V}}]}
\end{array}
$$

The approximants of a maximal fixpoint are defined dually:

$$
\begin{array}{rcl}
\|\nu Z^{<\alpha}.\phi\|^{\mathcal{T}}_{\mathcal{V}} & = & \bigcap_{\beta<\alpha} \|\nu Z^{\beta}.\phi\|^{\mathcal{T}}_{\mathcal{V}} \\
\|\nu Z^{\alpha}.\phi\|^{\mathcal{T}}_{\mathcal{V}} & = & \|\phi\|^{\mathcal{T}}_{\mathcal{V}[Z:=\|\nu Z^{<\alpha}.\phi\|^{\mathcal{T}}_{\mathcal{V}}]}
\end{array}
$$

Note that $\mu Z^{<0}.\phi \Leftrightarrow \mathrm{ff}$ and $\nu Z^{<0}.\phi \Leftrightarrow \mathrm{tt}$. By abuse of notation, we write Z^{α} or ϕ^{α} to mean $\substack{\mu \\ \nu} Z^{\alpha}.\,\phi$; of course this only makes sense when one knows which fixpoint and variable is meant.

We should remark here that most literature on the modal mu-calculus uses a slightly different definition, putting $\mu Z^0.\phi = \mathrm{ff}$, $\mu Z^{\alpha+1}.\phi = \phi(\mu Z^{\alpha}.\phi)$, and $\mu Z^{\lambda}.\phi = \bigcup_{\beta<\lambda} \mu Z^{\beta}.\phi$ for limit λ—which in effect is writing α for our $<\alpha$. That notation is inherited from set theory; its advantage is that a limit approximant is the limit of approximants. Our notation is inherited from more recent set theory; its advantages are that it sometimes reduces the number of trivial case distinctions in inductive proofs. However, the difference is not significant.

3.5 Examples

We have seen, both informally and in the formal semantics, the meaning of the fixpoint operators, and we have seen some simple examples of the modal mu-calculus translating CTL. We now consider some examples of modal mu-calculus formulae in their own right, which express properties one might meet in practice.

There is a well-known 'classification' [?] of basic properties into safety and liveness. In mu-calculus terms, it is not unreasonable to say that μ is liveness and ν is safety. Consider first simple ν formulae. For example:

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

is a relativized 'always' formula: 'P is true along every a-path'. Slightly more complex is the relativized 'while' formula

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

'on every a-path, P holds while Q fails'. Both formulae can be understood directly via the fixpoint construction, or via the idea of 'ν as looping': for example the second formula is true if either Q holds, or if P holds and wherever we go next (via a), the formula is true, and ..., and because the fixpoint is maximal, we can repeat forever. So in particular, if P is always true, and Q never holds, the formula is true.

μ formulae, in contrast, require something to happen, and thus are liveness properties. For example

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

is 'on all infinite length a-paths, P eventually holds'; and

$$\mu Z.Q \vee (P \wedge \langle a\rangle Z)$$

is 'on some a-path, P holds until Q holds (and Q *does* eventually hold)'. Again, these can be understood by 'μ as 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.

This level of complexity suffices to translate CTL, since we have $\mu Z.Q \vee (P \wedge \langle -\rangle Z)$ as a translation of $\exists[P\,\mathbf{U}\,Q]$, and $\mu Z.Q \vee (P \wedge [-]Z \wedge \langle -\rangle)$ as a translation of $\forall[P\,\mathbf{U}\,Q]$; and obviously we can nest formulae inside one another, such as

$$\nu Z.(\mu Y.P \vee \langle\rangle Y) \wedge [-]Z$$

'it is always possible that P will hold', or $\forall\mathbf{G}(\exists\mathbf{F}P)$. Equally obviously, we can write formulae with no CTL translation, such as

$$\mu Z.[a]\mathrm{ff} \vee \langle a\rangle\langle a\rangle Z$$

which asserts the existence of a maximal a-path of even length; a formula which is, incidentally, expressible in PDL. This is, however, a fairly simple extension; much more interesting is the power one gets from mixing fixpoints that depend on one another. Consider the formula

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

This formula is likely to stymie the average newcomer to the modal mu-calculus, but it has a simple meaning, which can be seen by using the slogans. $\mu Y....$ is true if $\nu Z....$ is true if $(P \wedge [a]Y) \vee (\neg P \wedge [a]Z)$, which is true if either P holds and at the next (a)-states we loop back to $\mu Y....$, or P fails, and at the next states we loop back to $\nu Z.....$ By the slogan 'μ means finitely', we can only loop through $\mu Y....$ finitely many times on any path, and hence P is true only finitely often on any path.

A variant on this theme illustrates the ease with which the modal mu-calculus can talk about actions, which from a traditional CCS viewpoint are all that one can observe. By similar reasoning to the above, one can see that

$$\mu Y.\nu Z.[a]Y \wedge [-a]Z$$

asserts that a process does only finitely many a actions, whatever else it does: it may even diverge. For example, consider the divergent value-passing CCS process

$$A(n) \stackrel{\mathrm{def}}{=} \tau.A(n+1) + \tau.B(n) \qquad B(n) = \textbf{if } n > 0 \textbf{ then } a.B(n-1).$$

$A(0)$ satisfies the formula. To see that this really is true, as the slogans suggest it is, we can examine the semantics. As this is an example of a nested fixpoint, let us consider the approximants. The first approximant of the outer minimal fixpoint is

$$\mu Y^0.... = \nu Z.[a](\mu Y^{<0}....) \wedge [-a]Z = \nu Z.[a]\mathrm{ff} \wedge [-a]Z.$$

We may also calculate this maximal fixpoint by approximants. Its first approximant is $\nu Z^0.... = [a]\mathrm{ff} \wedge [-a]\mathrm{tt}$, which is, by inspection, the processes $A(i)$ ($i \geq 0$) and $B(0)$. The next approximant is $\nu Z^1.... = [a]\mathrm{ff} \wedge [-a](\nu Z^{<1}....)$, which again by inspection is just $B(0)$; and here we stabilize. So we are now in a position to calculate the next outer approximant:

$$\mu Y^1.... = \nu Z.[a](\mu Y^{<1}....) \wedge [-a]Z = \nu Z.[a]\{B(0)\} \wedge [-a]Z.$$

The reader may compute this by approximants to obtain $B(0), B(1)$. Continuing, we obtain $\mu Y^i.... = \{\, B(j) \mid j \leq i \,\}$. The next approximant is where everything happens:

$$\mu Y^\omega.... = \nu Z.[a](\mu Y^{<\omega}....) \wedge [-a]Z = \nu Z.[a]\{B(i)\} \wedge [-a]Z.$$

A further calculation shows that here $\nu Z^0....$ is all processes, and so we stabilize; now the outer approximant chain has also converged at all processes.

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

$$\nu Y.\mu Z.\langle a\rangle Y \vee \langle\tau\rangle Z.$$

Applying the slogans, we see that this is true if either we can do an a and loop to Y, or do a τ, and loop to Z. We can only loop to Z finitely often, so we can only do finitely many consecutive τs; but if we do an a and loop back out to Y, we then re-enter the inner fixpoint, free to do another independent finitely many τs. The *really* enthusiastic reader might like to evaluate this formula by approximants on the modified set of processes given by putting $B(n) = \tau.A(2n) + \textbf{if } n > 0 \textbf{ then } a.B(n-1)$.

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 increases the complexity of model-checking—at least, as far as we know, although whether it really does is the remaining open problem about the modal mu-calculus. However, the practitioner who is alarmed by the thought of even more complex formulae, 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 formulae might appear. The first is if one has a front end working in CTL* (or another complex logic), and is translating into modal mu-calculus for the main engine. This is theoretically possible, by work of Mads Dam [15], 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*. Fairness properties, whether weak (an event continuously enabled must happen) or strong (an event enabled infinitely often must happen), can be written in the modal mu-calculus, and require mixing fixpoints. For example, to say that an action a is fairly treated means that there are no paths on which a is enabled infinitely often, but occurs only finitely often; this can be written as

$$\nu X.\mu Y.\nu Z.[a]X \wedge (\langle a\rangle \mathtt{tt} \Rightarrow [-a]Y) \wedge [-a]Z.$$

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 a is fairly treated', which is left as an exercise for the reader. In practice, such formulae are often actually equivalent to formulae of lower alternation depth, and in any case there are often better ways of handling fairness.

3.6 Modal mu-calculus as a process logic

So far, we have described the modal mu-calculus as a logic on labelled transition systems; but this is supposed to be a chapter about modal logics for process algebras. Of course, process algebras, or at least simple traditional process algebras, are equipped with an operational semantics that defines a labelled transition system, in which the states are usually pieces of syntax. The easy way to interpret any modal logic on a process algebra is just to use this generated transition system. However, in theory one's notion of process is not just a piece of syntax, but some *behaviour*, and one wishes to treat identically 'processes' that are behaviourally equivalent. In the CCS world view, we believe that processes that are strongly bisimilar are equivalent for all practical purposes—and most non-practical purposes—and in many situations we might treat observational congruence or weak bisimulation similarly. In more exotic situations we might view any of the hundreds of known equivalences as process equality. This raises the obvious question of how our logic behaves with respect to such equivalences, and what should we do if it behaves badly. The first answer is that the logic behaves well:

If two processes (i.e. states in transition systems) are strongly bisimilar, then they satisfy exactly the same modal mu-calculus formulae.

This is proved by an easy induction. In fact, for finite-branching systems (more precisely, image-finite systems) the converse is also true, so that modal mu-calculus (or indeed its small subset HML) characterizes strong bisimulation; this fails for arbitrary systems.

However, if we prefer, as we usually do, to work with a weaker equivalence, this of course is no longer true, since the modal mu-calculus as we have defined it treats τ just as any other action. There are several ways to deal with this. One is just to accept it, and when writing properties in the mu-calculus to be very careful to write explicitly a formula that does the desired thing with silent actions; this has obvious drawbacks. A more natural option, provided that whatever tool one is using supports it, is to work with the weak transition system, namely the transition system defined by $s \stackrel{a}{\Longrightarrow} t$ if $s \xrightarrow{\tau^*}\xrightarrow{a}\xrightarrow{\tau^*} t$. A compromise is to stick with the strong transition system, but to define derived modalities which take account of τ. For example, one can define $\langle\!\langle a\rangle\!\rangle\phi$ to mean $\mu Z.\langle\tau\rangle Z \vee \langle a\rangle \mu Z.\phi \vee \langle\tau\rangle Z$, which captures in the strong logic the diamond modality in the weak transition system.

Another issue that affects practice is the atomic propositions. We have presented the logic with arbitrary valuations for atomic propositions, but this is neither possible nor desirable in practice. Typically there will be some small language of atomic propositions; for example, in value-passing CCS one wants to have basic expressions over the parameters of a process. One might imagine that by restricting this language one could obtain better results on decidability etc.; although this is, loosely speaking, true in practice, in the sense that 'sensible' properties of 'sensible' systems can be expressed and often proved entirely within a 'sensible' language, it is not generally true in a mathematical sense, since the basic mu-calculus operators are sufficient to expose any undecidability that may be lurking in a system – for example, the halting property is just $\mu Z.[-]Z$.

3.7 Linear time mu-calculus

The modal mu-calculus is a pure branching time logic, although as we have seen in the examples it can express properties that are usually found in mixed branching/linear logics such as CTL*. One may also add fixpoint operators to linear time logic, to get a pure linear time mu-calculus. The modality in this case is a 'next' operator @, and the logic is interpreted not over states of a system, but over *runs* or maximal paths. The formula $@\phi$ is satisfied by the runs $\sigma = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots$ such that $a_1 = a$ and $s_1 \xrightarrow{a_2} \ldots$ satisfies ϕ. Adding negation, conjunction and maximal fixpoint to this operator gives the linear time mu-calculus.

Just as in the modal case, the linear mu-calculus encompasses most other linear time logics. The linear 'until' operator $\phi\, \mathbf{U}\, \psi$ is just $\mu Z.\psi \vee (\phi \wedge \bigcirc Z)$, and linear fairness operators can be expressed in a similar manner to their branching counterparts.

At first sight, the linear mu-calculus is much harder to handle than the modal logic: we are interpreting over runs, and even a finite-state system generates continuum-many runs. So we are taking fixpoints in a lattice of size 2^{2^ω}, and our approximants may, at first sight, not close until $(2^\omega)^+$! Of course, this doesn't actually happen, and in fact for a finite-state system fixpoints close at a finite approximant. However, this approximant may be large, and model-checking is exponential [**?**]. It is possible to give a direct procedure, and even a direct local procedure, to model-check the linear time mu-calculus [59, 10], but they are complicated, and for all but the smallest examples, automata-theoretic techniques are superior.

On the other hand, the linear mu-calculus is in some ways simpler than the modal mu-calculus. In particular, increasing alternation does not give increasing expressive power:

any formula is equivalent to one of alternation depth two. Indeed, any formula is equivalent to an ω-regular expression.

Generally speaking, the linear time mu-calculus has not offered the same advantages as the modal mu-calculus, and those who use a linear time logic use the traditional logic of ‘next’ and ‘until’, and its fragments.

4 Theory of modal mu-calculus

4.1 Fixpoint regeneration and the ‘fundamental semantic theorem’

In the informal description of the meaning of fixpoints, we used the idea of the dependency of s at ϕ on t at ψ. We now make this precise. Assume a structure $\mathcal{T}$, and a formula ϕ. Suppose that we annotate the states with sets of subformulae, such that the sets are locally consistent: that is, s is annotated with a conjunction iff it is annotated with both conjuncts; s is annotated with a disjunction iff it is annotated with at least one disjunct; if s is annotated with $[a]\psi$ (resp. $\langle a\rangle\psi$), then all (resp. at least one) a-successor is annotated with ψ; if s is annotated with a fixpoint or fixpoint variable, it is annotated with the body of the fixpoint. We call such an annotated structure a *quasi-model*.

A *choice function* f is a function which for every disjunctive subformula $\psi_1 \vee \psi_2$ and every state annotated with $\psi_1 \vee \psi_2$ chooses one disjunct $f(s, \psi_1 \vee \psi_2)$; and for every subformula $\langle a\rangle\psi$ and every state s annotated with $\langle a\rangle\psi$ chooses one a-successor $t = f(s, \langle a\rangle\psi)$ 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 state s that satisfies a subformula ψ are defined thus: $s@\psi_1 \wedge \psi_2 \succ s@\psi_i$ for $i = 1, 2$; $s@[a]\psi \succ t@\psi$ for every t such that $s \xrightarrow{a} t$; $s@\psi_1 \vee \psi_2 \succ s@f(s, \psi_1 \vee \psi_2)$; $s@\langle a\rangle\psi \succ f(s, \langle a\rangle\psi)@\psi$; $s@{}^{\mu}_{\nu}Z.\,\psi \succ s@\psi$; $s@Z \succ s@\psi$ where Z is bound by ${}^{\mu}_{\nu}Z.\,\psi$. 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 *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 mu-calculus can now be stated:

Theorem 1 *A well-founded pre-model is a model: in a well-founded pre-model, if s is annotated with ψ, then indeed $s \models \psi$.*

The theorem in this form is due to Streett and Emerson in [60], from which the term ‘well-founded pre-model’ is taken. Stirling and Walker [57, 58] presented 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 [11, 6] is the fundamental theorem, again with a slight relaxation on choice.

A converse is also true:

Theorem 2 *If in some structure $s \models \phi$, 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 ‘μ means finite looping’. To explain why it is true, we need to make a finer analysis of approximants.

Assume a structure $\mathcal{T}$, valuation $\mathcal{V}$, and formula ϕ in positive normal form. 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 $\mu Y_1.\langle a\rangle\mu Y_2.(P \vee Y_1) \vee \langle b\rangle Y_2$, 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. This idea is the notion of *signature*. A signature is a sequence $\sigma = \alpha_1, \ldots, \alpha_n$ of ordinals, such that the i least fixpoint will be interpreted by its α_ith 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 [60], the Fischer–Ladner closure of ϕ was used, rather than the set of subformulae. The closure is defined as for PDL (see section 2.1), with the addition of the rule that if ${}^{\mu}_{\nu}Z.\,\psi(Z) \in \mathrm{cl}(\phi)$, then $\psi({}^{\mu}_{\nu}Z.\,\psi) \in \mathrm{cl}(\phi)$. The signatures were defined by syntactically unfolding fixpoints, rather than by semantic approximants. In [58] 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, and has the advantage that it adapts easily to the modal equation system presentation of modal mu-calculus, which we shall see below. Indeed, it arises more naturally from that system.

Add to the language a countable set of *constants* $U, V, \ldots$. Constants will be defined to stand for maximal fixpoints or approximants 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 $\|\mu Y_i^{\alpha_i}.\psi_i'\|_\sigma$, 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 state s satisfies ψ with signature σ, written $s \models_\sigma \psi$, if $s \in \|\psi'\|_\sigma$, 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 s is accompanied by a signature – write $s@\psi[\sigma]$. 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 $s@Y_i$ it strictly decreases in the ith component and is unchanged in the

$1,\ldots,i-1$'th components. It can now be shown, by a slightly delicate but not too difficult induction, that if $s@\psi[\sigma]$, then $s \models_\sigma \psi$. Furthermore, given a well-founded pre-model, one can construct a locally consistent signature annotation—essentially, the Y_i component of σ in $s@\psi[\sigma]$ is the maximum 'number' (in the transfinite sense) of Y_i occurrences without meeting a higher variable in trails from $s@\psi$, and so on; the well-foundedness of the pre-model guarantees that this is well-defined. 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 [60], [58], [6].)

The converse is quite easy: given a model, annotate the states by the subformulae they satisfy; for $s@\psi$ assign the least σ such that $s \models_\sigma \psi$; 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.

4.2 The finite model property and decidability

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

Theorem 3 *The modal mu-calculus is decidable; that is, it is decidable whether a formula has some model.*

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

Theorem 4 *If a modal mu-calculus formula has a model, 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 (just unravel the original model), and has bounded branching degree (just cut off all the branches that are not actually required by some diamond subformula; this leaves at most (number of diamond subformulae) branches at each node). 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 section 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 unravelling 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 state s labelled by Z; let F be the set of formula annotating s. Now, if there is a state t below s that is also annotated by F, prune the tree at t, and identify t with s. Repeat for all Z labelled

states 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 fix-point 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 filtration technique we saw 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 we *meaning I* am not aware of any direct construction that gives the exponential upper bound of [60], which is in fact also the lower bound.

Is that what Kozen did in his proof? I don't have his paper with me.

4.3 Axiomatization

A related problem to decidability is the question of providing an axiomatization of the theory of the modal mu-calculus. In his original paper, Kozen presented the following axiomatization of the equational theory, where $\phi \le \psi$ means $\phi \vee \psi = \psi$, taking $\langle\rangle$ and μ as primitives and defining $[]$ and ν by duality:

1. axioms for Boolean algebras
2. $\langle a\rangle\phi \vee \langle a\rangle\psi = \langle a\rangle(\phi \vee \psi)$
3. $\langle a\rangle\phi \wedge [a]\psi \le \langle a\rangle(\phi \wedge \psi)$
4. $\langle a\rangle\mathrm{ff} = \mathrm{ff}$
5. $\phi(\mu X.\phi(X)) \le \mu X.\phi(X)$
6. $\frac{\phi(\psi)\le\psi}{\mu X.\phi(X)\le\psi}$

Axiom 5 is the axiom of fixed point induction, in dual form; rule 6 says that μ is indeed the least pre-fixed point.

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* formulae, 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 Igor Walukiewicz in [66], who established that Kozen's axiomatization is indeed complete. David Janin and Walukiewicz generalized 'aconjunctive' to the slightly misleadingly named *disjunctive formulae*, which are sufficient to express all mu-calculus properties—the proof of this is not straightforward. Walukiewicz was able to extend Kozen's proof to this more general notion—which was also not straightforward. Janin and Walukiewicz has also used disjunctive formulae to establish other interesting results [28].

4.4 Model-checking

Decidability and axiomatization are standard questions for logicians; but for the practioner, the important question is model-checking: given a system $\mathcal{T}$ with some initial state s and a formula ϕ, is it the case that $s \models \phi$? For the modal mu-calculus, the decidability of this question is obvious, for finite systems; the problem is, what is its complexity? There have been great technological advances in the model-checking of finite-state systems by the use of clever data structures such as BDDs. There have also been useful advances for fragments of the modal mu-calculus. However, there has been only one notable improvement in our knowledge of the complexity of the general problem, and a full answer to the question remains something that haunts our dreams—literally, for at least one of the authors!

To discuss the complexity of model-checking, we need to introduce the notion of *alternation depth* of a formula. We shall define this more precisely in the following subsection, which examines alternation. For the moment, it is sufficient to define it loosely as the maximum number of μ/ν alternations in a chain of nested fixpoints. In fact we usually add the proviso that the nesting is genuine interdependence of the 'infinitely often' kind rather than mere inclusion of closed subformulae of the 'always eventually' kind (see 3.5); but the proviso may be ignored without changing the essentials of the arguments.

Suppose that we simply 'model-check' by brute force calculation of the approximants. Let the size (number of states plus number of transitions) of the system be n, the length of the formula be m. Every fixpoint closes at, at worst, the nth approximant. Calculating the booleans takes time $O(n)$; calculating modalities takes time $O(n)$. For an innermost fixpoint, given a valuation of the free variables, we may need to evaluate the body n times to close. So the innermost fixpoint takes time $O(n \cdot (mn))$. Now the next fixpoint out may also need to be evaluated n times, each of which involves evaluating the inner fixpoint (and some non-fixpoint operators); and so that takes $O(n \cdot n \cdot (mn))$. And so on: so we end up with $O(m \cdot n^{d+1})$, where d is the depth of fixpoint operators.

Now it turns out that by some lattice theory due to Bekič, adjacent fixpoints of the same type, such as $\mu Y.\mu Z.\phi(Y,Z)$ are not much worse than one, owing to monotonicity: since $\|\mu Z.\phi(Y^{\alpha+1}, Z)\| \supseteq \|\mu Z.\phi(Y^{\alpha}, Z)\|$, when one evaluates the ith outer approximant, one need not start evaluating the inner fixpoint from 0, but from wherever one had reached on finishing the $(i-1)$th outer approximant. Alternatively, one can turn $\mu Y.\mu Z.\phi$ into a single fixpoint over a vector: $\mu(X,Y).\phi$. Thus we obtain the theorem of Emerson and Lei [20]:

Theorem 5 *The model-checking problem for a formula of size m and alternation depth d on a system of size n is $O(m \cdot n^{d+1})$.*

This theorem holds for the simple-minded notion of alternation, for the somewhat more refined version that Emerson and Lei used, and for the stronger notion of Niwiński [45].

So model-checking appears to be exponential in alternation depth. If, in the usual way, one defines the size of the problem to be the size of the system plus the size of the formula, this means that the problem seems to be exponential. Of course, it is a very well-behaved exponential, both theoretically, as by stratifying the formulae according to alternation depth one gets a chain of polynomial problems of increasing degree; and practically, because systems are typically large, formulae are typically small, and alternation depths are typically 1 or 2. However, it does not appear that the model-checking problem is polynomial.

On the other hand, it is readily apparent that the problem is in NP. This was first remarked by Emerson, Jutla and Sistla in [19], who used an automata-theoretic argument. However, it can be seen directly [56]: take a model, and guess the right choice function and annotation to make it a well-founded pre-model. Now to check that it really is a well-founded pre-model, we need to check local consistency of the annotation, which is polynomial (and in fact quadratic); and we need to check well-foundedness. But to check well-foundedness on a finite pre-model, it suffices to show that there are no bad loops in the dependencies: if a minimal fixpoint occurs infinitely often without a higher fixpoint appearing, then there must be some state that occurs twice (and indeed infinitely often) labelled with that fixpoint. To check this, we can do strongly connected component analysis: starting from the bottom up, for each minimal fixpoint, cut the dependency graph at all higher fixpoints, and then look for a strongly connected component involving the said minimal fixpoint. This analysis is polynomial, of low degree; and we do it for each fixpoint; so the whole check is polynomial.

Since model-checking is trivially closed under complementation, the problem is in NP $\cap$ co-NP. This places it in the very small class of such problems that are not known to be polynomial. Opinion is divided on the likely true complexity; some believe it to be polynomial, but although the P faction includes some most accomplished complexity theorists, no polynomial algorithm has yet been found, either for the mu-calculus itself, or for the stochastic pay-off games to which it can be reduced and which the complexity theorists are really interested in. Others, including exactly one of the authors, believe it to be non-polynomial; although this would make the world a more interesting place, this belief has the obvious disadvantage of being impossible to confirm definitely without a resolution of P = NP. Of course, even a proof of non-polynomiality subject to P $\neq$ NP would be highly interesting.

We mentioned that there has been one advance in the complexity of global model-checking. This is due to Long et al.[37]. Essentially, they observed that the monotonicity argument that we applied above to adjacent least fixpoints, can also be applied to a formula of shape $\mu Y.\nu Z.\phi(Y,Z)$; by a careful and quite subtle analysis, it can be shown that actually this requires quadratic, not cubic, time to compute. Unfortunately, this does not allow all alternation to be collapsed, but they showed that it generalizes to the extent of proving

Theorem 6 *Model-checking is $O(m \cdot n^{\lceil d/2 \rceil + 1})$.*

4.5 Alternation

We now look at the notion of alternation in more detail. As we have said, 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

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

is, using the brute-force model-checker above, 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

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

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 subformulae 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, $\mu X.\nu Y.[]Y \wedge \mu Z.[](X \vee Z)$ 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.

The alternation depth referred to in the complexity of model-checking is a measure of alternation that is symmetric in μ and ν. It is possible to give algorithms that compute the alternation depth of a formula [20, 1, 31], and this is how the notion was presented by Emerson and Lei. However, for our purposes it is easier to start from a definition of classes for formula, formalizing the idea of 'a $\mu/\nu/\mu$ formula' etc.; such a definition is analogous to the usual definition of quantifer alternation for predicate logic, an analogy which will be exploited later. This was how Niwiński [45] presented the notion of alternation, and we follow his presentation.

Assuming positive form, 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 $\Sigma_{n+1}^{N\mu}$ (resp. $\Pi_{n+1}^{N\mu}$, take $\Sigma_n^{N\mu} \cup \Pi_n^{N\mu}$, and close under the following rules:

1. if $\phi_1, \phi_2 \in \Sigma_{n+1}^{N\mu}$ (resp. $\Pi_{n+1}^{N\mu}$), then $\phi_1 \vee \phi_2, \phi_1 \wedge \phi_2, \langle a \rangle \phi_1, [a]\phi_1 \in \Sigma_{n+1}^{N\mu}$ (resp. $\Pi_{n+1}^{N\mu}$);

2. if $\phi \in \Sigma_{n+1}^{N\mu}$ (resp. $\Pi_{n+1}^{N\mu}$), then $\mu Z.\phi \in \Sigma_{n+1}^{N\mu}$ (resp. $\nu Z.\phi \in \Pi_{n+1}^{N\mu}$);

3. if $\phi(Z), \psi \in \Sigma_{n+1}^{N\mu}$ (resp. $\Pi_{n+1}^{N\mu}$), then $\phi(\psi) \in \Sigma_{n+1}^{N\mu}$ (resp. $\Pi_{n+1}^{N\mu}$), *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', we obtain the Emerson–Lei notion $\Sigma_n^{EL\mu}$. (We write just Σ_n^{μ} when the distinctions are 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 be the least n such that $\phi \in \Sigma_{n+1}^{\mu} \cap \Pi_{n+1}^{\mu}$. To make these definitions clear, consider the following examples:

- The 'always eventually' formula is $\Pi_2^{S\mu}$, but not $\Sigma_2^{S\mu}$, and so its simple alternation depth is 2. However, in the Emerson–Lei notion, it is also $\Sigma_2^{EL\mu}$, since $\nu Y.W \wedge \langle a \rangle Y$ is $\Pi_1^{EL\mu}$ and so $\Sigma_2^{EL\mu}$, and by substituting the closed $\Sigma_2^{EL\mu}$ (and in fact $\Sigma_1^{EL\mu}$) formula $\mu Z.P \vee \langle a \rangle Z$ for W we get 'always eventually' in $\Sigma_2^{EL\mu}$; hence its Emerson–Lei (and Niwiński) alternation depth is 1.

- The 'infinitely often' formula is Σ_2^{μ} but not Π_2^{μ}, in all three definitions, and so has alternation depth 2.

- The formula $\mu X.\nu Y.[]Y \wedge \mu Z.[](X \vee Z)$ is $\Sigma_3^{S\mu}$, but not $\Pi_3^{S\mu}$; it is also $\Sigma_3^{EL\mu}$ but not $\Pi_3^{EL\mu}$, since there are no closed subformulae to bring the substitution clause into play. However, in the Niwiński definition, it is actually $\Sigma_2^{N\mu}$: $\nu Y.[]Y \wedge W$ is $\Pi_1^{N\mu}$ and so $\Sigma_2^{N\mu}$; we can substitute the $\Sigma_1^{N\mu}$ formula $\mu Z.[](X \vee Z)$ for W without variable capture, and so $\nu Y.[]Y \wedge \mu Z.[](X \vee Z)$ is $\Sigma_2^{N\mu}$; and now we can add the outer fixpoint, still remaining in $\Sigma_2^{N\mu}$.

We have already discussed the role alternation depth plays in the complexity of model-checking. A natural question is whether the hierarchy of properties definable by Σ_n^{μ} formulae is actually a strict hierarchy, or whether the hierarchy collapses at some point so that no further alternation is needed. This problem remained open for a while; by 1990, it was known that $\Sigma_2^{N\mu} \neq \Pi_2^{N\mu}$ [2]. No further advance was made until 1996, when the strictness of the hierarchy was established independently by Giacomo Lenzi and Bradfield:

Theorem 7 *For every n, there is a formula $\phi \in \Sigma_n^{\mu}$ which is not equivalent to any Π_n^{μ} formula.*

Bradfield established this for $\Sigma_n^{N\mu}$, which implies the result for the other two notions; Lenzi established it for $\Sigma_n^{EL\mu}$, but his proof almost certainly works for $\Sigma_n^{N\mu}$ also.

Lenzi's proof is technically complex *this is code for: is there actually anybody out there who understands it?*, 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 [9]. The trick now is to transfer this hierarchy to the modal mu-calculus. Simply by writing down the semantics, it is clear (give or take some work to deal with the more complex notions of alternation) that if one takes a recursively presented transition system and codes it into the integers, then for a modal formula $\phi \in \Sigma_n^{\mu}$, its denotation $\|\phi\|$ is describable by an arithmetic Σ_n^{μ} 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 can be 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 formulae. Bradfield's examples are:

Theorem 8 *The Σ^μ_n formula*

$$\mu X_n.\nu X_{n-1}.\ldots.\mu X_1.[c]X_1 \vee \langle x_1\rangle X_1 \vee \ldots \vee \langle x_n\rangle X_n$$

is not equivalent to any Π^μ_n formula.

In a later section, we shall see another example of a hard formula, which arises naturally from the game-theoretic understanding of mu-calculus.

5 More on model-checking

In the last section, we saw the approximant approach to model-checking finite systems. In this section, we shall look at some other techniques that have been proposed, and at some of the extensions to deal with classes of infinite processes.

5.1 Local model-checking ...

The approximant calculation of model-checking is inherently global; calculating a fixpoint approximant involves applying a function over the state space. Stirling and Walker in [57] advocated the use of *local* techniques, that is, techniques where to determine the truth of $s \models \phi$, one looks at the immediate neighbourhood of s as required by the formula; if there are fixpoints, one will of course move further and further away from s, but one can stop as soon as the truth or falsity of ϕ at s is established, rather than working with the whole state space. The formulation they adopted was inspired by tableau methods in logic. In outline, suppose we want to check $s \models \phi$. We start with a *sequent* $s \vdash \phi$, and then apply rules according to the structure of ϕ. For example, if $\phi = \phi_1 \wedge \phi_2$, there is an $\wedge$-rule which is:

$$\frac{s \vdash \phi_1 \wedge \phi_2}{s \vdash \phi_1 \quad s \vdash \phi_2}$$

and if we get to a sequent $s \vdash \langle a\rangle\psi$, we apply the $\langle\rangle$-rule

$$\frac{s \vdash \langle a\rangle\psi}{t \vdash \psi} \quad \text{for any } t \text{ s.t. } s \xrightarrow{a} t$$

and if we reach $s \vdash Z$, where Z is bound by ${}^{\mu}_{\nu}Z.\,\psi$, we apply the fixpoint rule[5]

$$\frac{s \vdash Z}{s \vdash \psi}$$

Thus we build up (or rather down, since the rules grow downwards) a goal-directed tree of sequents to be established. The question is, when do we stop? and how do we know we've succeeded? Obviously we stop on an a-modality if a state has no a-successors, in which case $s \vdash [a]\psi$ succeeds and $s \vdash \langle a\rangle\psi$ fails; and we stop at $s \vdash P$, when we look s up in the proposition valuation to determine success. The key is the treatment of fixpoints. If we get to $s \vdash {}^{\mu}_{\nu}Z.\,\phi$, and we have already seen the same sequent higher up in the tree, and we have not passed through a higher fixpoint since then, we stop; the sequent is successful for ν and unsuccessful for μ. Once we have a finite tree, we propagate success and failure back up to the root via the rules.

Stirling and Walker showed that this algorithm was sound and complete for finite systems. The proof of this fact was another development of the fundamental semantic theorem, for finite models. Indeed, it is probably obvious to the reader (at least the reader who has read section 4.1) that the construction of a successful tableau gives a well-founded pre-model, and that a well-founded pre-model is a tableau. This statement needs to be interpreted carefully: in tableau construction, one can meet a sequent $s \vdash \langle a\rangle\psi$ many times, and choose a different successor each time; so strictly speaking the pre-model is the tableau itself, which is an unravelling of the original system. Alternatively, one can say that a tableau constructs a well-founded quasi-model for the original system, where 'well-founded' is interpreted in the sense of alternating parity automata. Because of this, the soundness and completeness proofs for this system, and its infinite generalization, require a little more care than the simpler version used in the fundamental semantic theorem, but the differences are technical rather than essential.

5.2 ...for infinite systems

Apart from their locality, another feature of the goal-directed tableau approach is that it can be generalized to use symbolic methods, so allowing the proof of properties of infinite systems (or just large systems). The idea is to use sets of states in sequents, rather than single states; then in the $\vee$ and $\langle\rangle$-rules, an explicit choice function is given; termination happens when we construct a sequent $S \vdash Z$ such that we have already seen a sequent $S' \vdash Z$ for $S' \supseteq S$, with no higher fixpoint intervening; and success involves giving a proof that all the dependency trails are well-founded. This latter is typically done by defining some well-founded measure on processes, and showing that it decreases as one passes

[5] This is an abbreviated presentation. In the original [57] system, fixpoints were first replaced by constants, and then unfolded. This allowed the termination condition to be expressed more simply. Also, the $\vee$-rule and $\langle\rangle$-rule actually had all possible successors with a disjunctive success criterion, rather than choosing one successor. This corresponds to using alternating automata, or to constructing quasi-models rather than pre-models.

through minimal fixpoints, and doesn't increase elsewhere. The canonical such measure is the signature, as described in 4.1, but simpler measures may suffice. Thus an infinite state tableau constructs a finite presentation of a well-founded quasi- or pre-model. Details may be found in [6, 11], and an example of a Petri net implementation in [7].

5.3 Decidable model-checking for infinite processes

The infinite-state tableau system just described makes no claims for decidability: it allows arbitrary mathematical reasoning to be used in establishing the termination and success conditions. An obvious question is to ask what can be done for infinite processes while retaining decidability; the answer is quite limited.

Clearly model-checking is undecidable for any Turing-powerful class of processes, since $\mu Z.[]Z$ expresses the halting property. It turns out to be easy to code halting in the modal mu-calculus, or even in small fragments of linear temporal logic, for many classes that are not Turing-powerful by themselves. For example, if we try to model a register machine as a Petri net, we need to test for zero, which nets cannot do; but if d is a transition that can fire when a particular place is non-empty, then $[d]\mathrm{ff}$ expresses the emptiness of that place, and so one can write a mu-calculus formula which says 'the net halts, if we look at those "honest" computations where a branch-on-zero is only taken when the corresponding counter really is zero'. Similarly, it is unnecessary for the registers actually to synchronize with the finite-state control, as we can encode the synchronization into the formula. The upshot of this is that model-checking, even with just one fixpoint, is undecidable for BPP, Basic Parallel Processes—a result which might be surprising, given the decidability of bisimulation for BPP.

On the other hand, for BPA, Basic Process Algebra, and for its generalization pushdown processes, model-checking is decidable. This follows from a classical result of Muller and Schupp [43] on the monadic second-order theory of pushdown graphs; but it has been shown by a direct and elegant argument by Burkart and Steffen [12]. They view pushdown processes as 'higher-order' transition graphs. For example, the BPA process system given by $A = aBA + c$ and $B = bBA + c$ is viewed as a system where the A process does an a transition followed by a B transition, understood as a recursive call to B, and returns to its initial state (if B terminates), or does a c to terminate. Now, if we are interested in a particular formula ϕ, each transition, including the 'higher-order' transitions, is viewed as a predicate transformer on the subformulae of ϕ: assuming that some subformula hold at the terminal state of the process, which formulae hold at the initial state? Obviously there are only finitely many possibilities, and the result can be calculated by an approximant-style procedure. If one now plugs into the A transformer the formulae that are true at a real terminal state, one obtains the formulae true of A.

Further detail on all these results can be found in the chapter by Burkart and Caucal later in this handbook.

5.4 Equation systems and efficient local model-checking

The finite-state tableau local model-checking method of [57] is easy to understand, but not at all efficient in the worst case; as originally presented it has double exponential complexity. Although this can be improved, by using a less simplistic termination condition, and by allowing sharing between subtableaux (in the tableau recycling of Angelika Mader [38]), it does not approach the efficiency of global model-checking. Recently, efficient local algorithms have been developed, using a version of the mu-calculus that we have already mentioned in passing, and will now describe.

Recall that in section 3.1 we presented fixpoints as modal equations. We have also seen mention of vectorial fixpoints, allowing the collapse of adjacent fixpoints of the same sign into one. Combining these ideas, we say a *modal equation system* is a set $\{ X_{ij} = \phi_{ij} \mid 1 \leq i \leq m, 1 \leq j \leq n_i \}$ of equations, where each ϕ_{ij} is a modal (no fixpoints) formula in the variables X_{ij}; for each i, the set $\{ X_{ij} = \phi_{ij} \mid 1 \leq j \leq n_i \}$ is a *block*, and each block is either maximal or minimal; say, wlog, that odd blocks are minimal and even blocks are maximal. There is no semantic ordering within a block, but the set of blocks is partially ordered; the ordering $1, \ldots, m$ of block indices is a linearization of this partial order. The semantics of such a system is given by treating each block as a vectorial fixpoint of the appropriate sign, with the nesting of fixpoints given by the partial ordering on blocks. By the obvious replacement of higher-numbered right-hand side variables by their defining equations, a modal equation system can be turned into a modal mu-calculus formula defining X_{11}.

Modal equation systems are a concise way of representing properties, as the direct use of variables referring to equations instead of explicit fixpoint formulae allows sharing between equations. For example, if $X_{11} = \phi_{11}$ and $X_{12} = \phi_{12}$ both refer to $X_{21} = \phi_{21}$, then to write the system in linear form we have to write the text of ϕ_{21} twice, once in each ϕ_{1i}; and so one can obtain an exponential (in the number of blocks) blowup by going from equation systems to normal modal mu-calculus. However, results about the modal mu-calculus in its linear form normally transfer without cost (save in intellectual effort) to equation systems.

A *Boolean equation system* (*BES*) is similar, except that the variables are just booleans, not states, and the equations contain only boolean operators. A modal equation system together with a modal structure can be turned into a boolean equation system, by treating a (state,modal variable) pair as a boolean variable. Perhaps surprisingly, this is sometimes worth doing: BES's have the advantage that they strip away the modal operators and the artificial linearization imposed by text, laying bare the problem of computing alternating fixpoints. BES's have been studied by, among others, Bart Vergauwen and Johan Lewi, and Angelika Mader [39]. Vergauwen and Lewi claimed an efficient local algorithm [65], but it is apparently buggy, and is also hard to understand. In 1998, Lui, Ramakrishnan and Smolka [36] presented a BES-based local model-checking algorithm that is quite short, relatively simple, and of similar efficiency to global model-checking.

6 Automata, games and logic

There are other formalisms that are intimately related to fixpoint logics, and in this section we shall give an overview of automata and games, which are the main examples. The relationship benefits both parties: the modal mu-calculus provides simple proofs of some automata-theoretic results, and automata theory is used in the proof of modal mu-calculus results.

6.1 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 *non-deterministic* 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 at each non-deterministic 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 non-deterministic tree automaton, the transition is chosen non-deterministically, 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 conditions as follows: instead of saying just that

every path through the run accepts, we can say that the subrun rooted at a node n is accepted if some boolean combination, depending on the state, $f(\{T_i\})$ holds, where T_i is the statement that the subrun rooted at n's ith 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, as we shall see later, but one can think of it in mu-calculus terms as being: there must exist a choice function which for each node selects a satisfying assignment for $f(\{T_i\})$, and thus selects some children; then every path through the selected nodes must meet the Rabin etc. acceptance condition.

If the boolean combination is just conjunction, $f(\{T_i\}) = \bigwedge_i T_i$, 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 non-determinism. It is probably not much of a surprise that:

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

The equivalence is simple: a diamond modality corresponds to a node 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 so 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. As a corollary, we have

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

The theory of automata on infinite objects is highly developed. Wolfgang Thomas [63] provides a survey of the entire area, and Niwiński's [46] 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 modal mu-calculus.

- In Rabin's original paper, one of the hardest lemmas was proving that Rabin automata are closed under complementation. Given the above, it is now 'obvious', since modal mu-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, a tree that is the unravelling of a finite system. This gives the finite model property of the modal mu-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 linear time mu-calculus formulae.*

The fact that Büchi and parity automata are equivalent on words is the automata-theoretic analogue of the fact that the fixpoint alternation hierarchy collapses in linear-time mu-calculus. On trees, alternating Büchi automata correspond to Π_2^μ modal mu-calculus formulae.

6.2 Monadic second-order logic

To continue the story, we bring in monadic second-order logic, studied by Rabin in his original paper. SnS is the monadic second-order logic of the n-ary tree, so that elements are nodes of the tree, the n successor relations are in the logic, first-order quantification over nodes is allowed, and quantification over sets, but not relations, of nodes is allowed. Rabin showed that SnS is equivalent to Rabin automata. It follows that on trees, SnS is equivalent to modal mu-calculus. This is a very curious result, since at first sight, SnS offers much more power. (It is obvious that mu-calculus is a sublogic of SnS since the definitions of greatest and least fixed point are monadic second-order definitions.) For one thing, we can talk about ancestors and cousins of nodes; for another, arbitrary second-order quantification looks more powerful than just fixpoints. But in fact neither of these gives more power. *Explaining the first is easy; I would love to explain the second in a couple of sentences! How about: roughly speaking, asserting the existence of a set of nodes in SnS can be coded into mu-calculus by using a least fixpoint to search forward for the members of the set. Does that make any sense?*

If we consider transition systems rather than trees, David Janin and Igor Walukiewicz have recently developed appropriate generalizations of this theory. We already know that modal mu-calculus respects bisimulation; it now turns out that

Theorem 13 *A formula in the monadic second-order language of transition systems is expressible in modal mu-calculus exactly if it respects bisimulation.*

this stuff needs some thought. I'm not sure there's a way of explaining it without the messy detail, which I don't understand.

6.3 Games

Games have played an important role in many areas of mathematics, but especially in logic. The so-called Ehrenfeucht–Fraïssé *check spelling* 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 mu-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 mu-calculus problems assist in exploring the close relationships with finite model theory.

A parity game is played by two players, Abelard and Eloise[6]. An arena for the game is a graph, in which each node is labelled as either an Abelard or Eloise node, and also labelled with a rank, as in parity automata. A play of the game consists of a sequence of moves in which the appropriate player chooses a successor of the current node; Eloise wins if the play ends in Abelard getting stuck, or if the play is infinite and the highest rank occurring infinitely often is even. (It may be convenient to designate certain nodes as immediate Abelard/Eloise wins, for example to deal with atomic propositions in the mu-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.

It should be obvious that, for example, modal mu-calculus model-checking can be viewed as a parity game: the graph (referred to as the game graph in [56]) comprises pairs of states and formulae, with transitions given by the dependencies (that is, it is a quasi-model, in the terminology of section ???). Abelard owns the conjunction, box, and maximal fixpoint nodes; Eloise the others; the ranks are assigned as for parity automata. The statement 'Eloise has a winning strategy' is equivalent to the statement that the corresponding alternating parity automaton has an accepting run is equivalent to the statement that the structure satisfies the formula. Equipping the structure with a choice function corresponds to giving Eloise a memoryless strategy; and so by the fundamental theorem and duality, a player has a winning strategy iff they have a memoryless winning strategy.

parity game formula; games in model-checking; abstract games

7 Extensions

Still to be written; just a quick glance at some non-interleaving logics

[6] Game presentations are plagued by the variety of names for the players. In Ehrenfeucht–Fraïssé games, they are traditionally called Duplicator and Spoiler, but the meaning of these names does not transfer happily to all situations. Other possibilities are Player and Opponent, or Player 0 and Player 1, or Player I and Player II. Our choice has the advantage of being formally meaningless, but with a mnemonic for remembering which player is which in applications: ∀belard and ∃loise.

References

[1] H. R. Andersen, *Verification of Temporal Properties of Concurrent Systems*, DAIMI PB – 445, Computer Science Dept, Aarhus University, 1993.

[2] A. Arnold and D. Niwinski, Fixed point characterization of Büchi automata on infinite trees, *J. Inf. Process. Cybern.* **EIK 26** (1990) 451–459.

[3] J. W. de Bakker, *Mathematical Theory of Program Correctness*. Prentice-Hall, Englewood Cliffs, NJ (1980).

[4] B. Banieqbal and H. Barringer, Temporal logic with fixed points, in *Temporal Logic in Specification*, LNCS **398** (Springer-Verlag, Berlin, 1989) 62–74.

[5] H. Barringer, R. Kuiper and A. Pnueli, Now you may compose temporal logic specifications. *Proc. 16th ACM Symp. on Theory of Computing* 51–63 (1984).

[6] J. C. Bradfield, *Verifying Temporal Properties of Systems* (Birkhäuser, Boston, 1991).

[7] A proof assistant for symbolic model-checking, *Proc. CAV '92*. (LNCS **663** 316–329 (1993).

[8] J. C. Bradfield, The modal mu-calculus alternation hierarchy is strict, in: U. Montanari and V. Sassone, eds., *Proc. CONCUR '96*, LNCS **1119** (Springer, Berlin, 1996) 233–246.

[9] J. C. Bradfield, Simplifying the modal mu-calculus alternation hierarchy, in: M. Morvan, C. Meinel and D. Krob, eds., *Proc. STACS 98*, LNCS **1373** (Springer, Berlin, 1998) 39–49.

[10] J. C. Bradfield, J. Esparza and A. Mader, An effective tableau system for the linear time mu-calculus, *Proc. ICALP '96* (LNCS 1099) (1996).

[11] J. C. Bradfield, Local model checking for infinite state spaces, *Theoret. Comput. Sci.* **96** 157–174, 1992.

[12] O. Burkart and B. Steffen, Model checking for context-free processes. *Proc. CONCUR '92*, LNCS 630, 123–137 (1992).

[13] E. M. Clarke and E. A. Emerson, Design and synthesis of synchronization skeletons using branching time temporal logic. LNCS 131 52–71 (1981).

[14] E. M. Clarke, E. A. Emerson and A. P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems* **8** 244–263 (1986).

[15] M. Dam, CTL* and ECTL* as Fragments of the Modal mu-Calculus, *Theoret. Comput. Sci.* **126** 77–96 (1994).

[16] E. Allen Emerson, Temporal and Modal Logic. Handbook of Theoretical Computer Science, Volume B., J. van Leeuwen Editor, Elsevier, 1990.

[17] Emerson, E, and Clarke, E. Characterizing correctness properties of parallel programs using fixpoints. *Lecture Notes in Computer Science*, **85**, 169-181 (1980).

[18] Emerson, E., and Jutla, C. (1991). Tree automata, mu-calculus and determinacy. In *Proc. 32nd IEEE Foundations of Computer Science.*

[19] Emerson, E., Jutla, C., and Sistla, A. On model checking for fragments of μ-calculus, *Lecture Notes in Computer Science*, **697**, 385-396 (1993).

[20] E. A. Emerson and C.-L. Lei, Efficient model checking in fragments of the propositional mu-calculus, in: *Proc. 1st LICS* (IEEE, Los Alamitos, CA, 1986) 267–278.

[21] M. J. Fischer and R. E. Ladner, Propositional dynamic logic of regular programs. *J. Computer and System Science* **18** 194–211 (1979).

[22] R. Floyd, Assigning meanings to programs. In J. T. Schwartz, *Mathematical Aspects of Computer Science* 19–32. American Mathematical Society, Providence (1967).

[23] Y. Gurevich and L. Harrington, Trees, Automata and Games, in: *Proceedings of the 14th Annual ACM Symposium on Theory of Computing* (San Francisco, CA, 1982), 60–65, ACM, New York, 1982.

[24] D. Harel, A. Pnueli and J. Stavi, Propositional dynamic logic of context-free programs. *Proc. 22nd Symp. on Foundations of Computer Science* 310–321. Nashville, Tennessee (1981).

[25] M. Hennessy and R. Milner, On observing nondeterminism and concurrency, *LNCS* **85**, 295-309 (1980).

[26] C. A. R. Hoare, An axiomatic basis for computer programming. *Comm. of the ACM* **12** 576–580 (1969).

[27] H. Hüttel, *SnS* can be modally characterized. *Theoret. Comput. Sci.* **74** 239–248 (1990).

[28] D. Janin and I. Walukiewicz, Automata for the μ-calculus and related results, in *Proc. MFCS '95*, LNCS **969** (Springer-Verlag, Berlin, 1995) 552–562.

[29] D. Janin and I. Walukiewicz, On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic, in *Proc. CONCUR '96*, LNCS **1119** (Springer-Verlag, Berlin, 1996) 263–277.

[30] R. Kaivola, On modal mu-calculus and Büchi tree automata, *Inf. Proc. Letters* **54** (1995) 17–22.

[31] R. Kaivola, *Using Automata to Characterise Fixpoint Temporal Logics*, Ph.D. thesis, University of Edinburgh (1997).

[32] D. Kozen, Results on the propositional mu-calculus, *Theoret. Comput. Sci.* **27** (1983) 333–354.

[33] D. Kozen, A finite model theorem for the propositional μ-calculus, *Studia Logica* **47** (1988) 233–241.

[34] D. Kozen and R. Parikh, An elementary proof of the completeness of PDL. *Theoret. Comput. Sci.* **14** 113–118 (1981).

[35] G. Lenzi, A hierarchy theorem for the mu-calculus, in: F. Meyer auf der Heide and B. Monien, eds., *Proc. ICALP '96*, LNCS **1099** (Springer, Berlin, 1996) 87–109.

[36] X. Liu, C.R. Ramakrishnan, and S.A. Smolka, Fully local and efficient evaluation of alternating fixed points. *Proc. TACAS '98*, LNCS (1998).

[37] D. Long, A. Browne, E. Clarke, S. Jha and W. Marrero, An improved algorithm for the evaluation of fixpoint expressions, in: D. L. Dill, ed., *Proc. CAV '94*, LNCS **818** (Springer, Berlin, 1994) 338–350.

[38] A. Mader, Tableau recycling, *Proc. CAV '92*, LNCS **663** (1992).

[39] Mader, A. *Verification of modal properties using boolean equation systems.* PhD Thesis, Technical University of Munich (1997).

[40] Z. Manna and A. Pnueli, Formalization of properties of recursively defined functions. *Proc. ACM Symp. on Theory of Computing* 201–210 (1969).

[41] Z. Manna and A. Pnueli, How to cook a temporal proof system for your pet language. *Proc. 10th ACM Symp. on Principles of Programming Languages* 141–154 (1983).

[42] Milner, R. *A Calculus of Communicating Systems.* Lecture Notes in Computer Science **92**.

[43] D. Muller and P. Schupp, The Theory of Ends, Pushdown Automata and Second Order Logic. *Theoret. Comput. Sci.* **37** 51–75 (1985).

[44] P. Niebert. A ν-calculus with local views for systems of sequential agents. *Proc. MFCS '95* (LNCS 969) 563–573. Springer-Verlag (1995).

[45] D. Niwiński, On fixed point clones, in: L. Kott, ed., *Proc. 13th ICALP*, LNCS **226** (Springer, Berlin, 1986) 464–473.

[46] D. Niwiński, Fixed point characterization of infinite behavior of finite state systems. *Theoret. Comput. Sci.* **189** (1997) 1–69.

[47] Park, D. Fixpoint induction and proofs of program properties. *Machine Intelligence*, **5**, 59-78, Edinburgh University Press (1969).

[48] W. Penczek, Temporal logics for trace systems, *Int. J. Found. Comput. Sci.* **4**(1) 31–67 (1993).

[49] A. Pnueli, Temporal semantics of concurrent programs. *Theoret. Comput. Sci.* **27** 333–354 (1983).

[50] V. Pratt, Semantical considerations of Floyd–Hoare logic. *Proc. 1th IEEE Symp. on Foundations of Computer Science* 109–121 (1976).

[51] Pratt, V. A decidable mu-calculus, *22nd IEEE Symposium on Foundations of Computer Science*, 421-427 (1982).

[52] Rabin, M. Decidability of second-order theories and automata on infinite trees. *Transactions of American Mathematical Society*, **141**, 1-35 (1969).

[53] M. O. Rabin, Weakly definable relations and special automata, in: Y. Bar-Hillel (ed.), *Mathematical Logic and Foundations of Set Theory* (North-Holland, Amsterdam, 1970) 1–23.

[54] A. P. Sistla and E. M. Clarke, Complexity of propositional temporal logics. *J. ACM* **32** 733–749 (1986).

[55] C. P. Stirling, Modal and temporal logics, in: S. Abramsky, D. Gabbay and T. Maibaum, eds., *Handbook of Logic in Computer Science*, Vol. 2 (Oxford University Press, 1991) 477–563.

[56] C. Stirling, Local model checking games, in: I. Lee and S. A. Smolka, eds., *Proc. Concur '95*, LNCS **962** (Springer, Berlin, 1995) 1–11.

[57] C. P. Stirling and D. J. Walker, Local model checking in the modal mu-calculus. *Proc. International Joint Conference on Theory and Practice of Software Development* 369–382. LNCS 351 (1989).

[58] C. Stirling and D. Walker, Local model checking in the modal mu-calculus, *Theor. Comput. Sci.* **89**, 161–177. (1991).

[59] C. Stirling and D. Walker, CCS, liveness, and local model checking in the linear time mu-calculus, *Proc. First International Workshop on Automatic Verification Methods for Finite State Systems*, LNCS **407** 166–178. (1990).

[60] R. S. Streett and E. A. Emerson, An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation* **81** (1989) 249–264.

[61] R. Streett, Propositional dynamic logic of looping and converse. *Proc. 13th Symp. on Theory of Computing* 375–383. Milwaukee, Wisconsin (1981).

[62] P. S. Thiagarajan, A trace based extension of linear time temporal logic. *Proc. 9th LICS* 438–447. IEEE (1994).

[63] W. Thomas, Languages, Automata, and Logic, in: *Handbook of Formal Language Theory* (G. Rozenberg, A. Salomaa, Eds.),Vol. III, Springer-Verlag , New York, pp. 389-455 (1998).

[64] M. Y. Vardi, A temporal fixpoint calculus, *Proc. 15th PoPL*, 250–259 (1988).

[65] B. Vergauwen and J. Lewi, Efficient local correctness checking for single and alternating boolean equation systems. *Proc. ICALP '94*, LNCS **820** 302–315 (1994).

[66] I. Walukiewicz, Completeness of Kozen's axiomatisation of the propositional μ-calculus. *Proc. 10th LICS*. IEEE (1995).