Modal fixpoint logic: some model theoretic questions
Fontaine, G.M.M.

Citation for published version (APA):
The $\mu$-calculus is an extension of modal logic with least and greatest fixpoint operators. Modal logic was originally developed by philosophers in the beginning of the 20th century [BRV01]. It aimed at combining the concepts of possibility and necessity with propositional logic. In the 1950s, the possible world semantics was introduced (see for instance [BRV01, BS84]) and since then, modal logic has proved to be an appealing language to reason about transition systems. In addition to its philosophical motivations, modal logic appears to be of interest in many other areas: essentially in any area that uses relational models as representation means. Examples include artificial intelligence, economics, linguistics and computer science.

In computer science, labelled transition systems are used to represent processes or programs. The nodes of the transition systems model the possible states of the process, whereas the edges represent the possible transitions from one state to another. The label of a given node carries all the local information about the node. Within that perspective, logic seems to be a natural tool to describe the properties of programs. This approach turned out to be particularly useful for specification and verification purposes.

Verification is concerned with correctness of programs. More specifically, given a program represented by a labeled transition system and a formula (called the specification), representing the intended behavior of the program, we want to check whether the formula holds in the transition system. This is nothing but the model checking problem for the logic used as a specification language.

In order to reason about programs, especially non-terminating ones, standard modal logic lacks expressive power. Usual types of correctness properties that one would like to formulate are safety (“nothing bad ever happens”) or fairness (“something good eventually happens”). Typically, such types of properties are expressed using a recursive definition (nothing bad ever happens if nothing bad happens now and it is the case that for the next states nothing bad ever happens). So it seems reasonable to enrich modal logic with operators capturing some form
Chapter 1. Introduction

of recursive principle. At the end of the 1970s, Amir Pnueli [Pnu77] argued that linear temporal logic (LTL), which is obtained by restricting to models based on the natural numbers and by adding the “until” operator to modal logic, could be a useful formalism in that respect. Since then, other temporal logics have been introduced, the most famous ones being computation tree logic [CE81] (CTL) and CTL* [EL86], and are considered as appropriate specification languages.

Around the same time, Vaughan Pratt [Pra76] and Andrzej Salwicki [Sal70] independently introduced Dynamic Logic. The basic idea of Dynamic Logic is to associate a modality $[\theta]$ with each program $\theta$; the intuitive meaning of a formula $[\theta]\phi$ is that $\phi$ holds in all states reachable after an execution of $\theta$. In 1977, a propositional version of Dynamic Logic (PDL) was introduced by Michael Fischer and Richard Ladner [FL79]. One disadvantage of Dynamic Logic is that unlike temporal logics, it is not adequate for modeling non-terminating programs. Extensions of PDL that can capture some specific infinite behaviors (see for instance [Har84]) have been studied by Robert S. Streett [Str81, Str82] (Delta-PDL), David Harel and Vaughan Pratt [HP78] (PDL with a loop construct).

Fixpoint logics are formalisms that can deal with both non-terminating behavior and recursion in its most general form. The basic idea of fixpoint logics is to explicitly add operators that allow us to consider solutions of an equation of the form $f(x) = x$. For example, safety is a solution of the equation “$x \leftrightarrow (\text{nothing bad happens now} \land \text{for all successors}, x)$”.

The first logic that was extended by means of fixpoint operators was first-order logic [Mos74]. The initial purpose was to establish a generalized recursion theory. In the context of semantics of programming languages, the use of fixpoints to enrich first-order logic goes back to Dana Scott, Jaco de Bakker [SDB69, Bak80] and David Park [Par69]. However, this required the development of a complex mathematical theory. A few years later, arose the idea of considering fixpoint extensions of modal logic. The most successful logic that came out of this approach is the $\mu$-calculus. Works of of E. Allen Emerson, Edmund Clarke [EC80], David Park [Par80] and Vaughan Pratt [Pra81] prefigured the actual definition of the $\mu$-calculus which was given in 1983 by Dexter Kozen [Koz83].

The $\mu$-calculus is obtained by adding the least fixpoint operator $\mu x$ and its dual, the greatest fixpoint operator $\nu x$, to the standard syntax for modal logic. Intuitively, the formula $\mu x.\phi(x)$ is the smallest solution of the equation $x \leftrightarrow \phi(x)$. Similarly, $\nu x.\phi(x)$ is the biggest solution of this equation.

Not surprisingly, adding fixpoint operators to modal logic results in a significant increase of the expressive power. Most temporal logics (including LTL, CTL and CTL*) can be defined in terms of the $\mu$-calculus [Dam94, BC96]. In fact, these logics usually fall inside to a rather small syntactic fragment of the $\mu$-calculus (the fragment of alternation depth at most 2).

Moreover, on binary trees, it follows from various results [Rab69, EJ91, Niw88, Niw97] that the $\mu$-calculus is equivalent to monadic second-order logic (MSO).
**MSO** is an extension of first-order logic, which allows quantification over subsets of the domain. It is also one of the most expressive logics that is known to be decidable on trees, whether they are binary or unranked (that is, there is no restriction on the number a successors of a node). Hence, it is not surprising that most specification languages are fragments of **MSO**. This means that on binary trees, the $\mu$-calculus subsumes most specification languages.

On arbitrary structures, it is easy to see that the $\mu$-calculus is a proper fragment of **MSO**. A key result concerning the expressive power of the $\mu$-calculus is the Janin-Walukiewicz theorem [JW96]: an **MSO** formula $\varphi$ is equivalent to a $\mu$-formula iff $\varphi$ is invariant under bisimulation. Bisimulations are used to formalize the notion of behavioral equivalence. The idea is that when specifying behaviors, one is interested in the behavior of programs rather than the programs themselves. Hence, a specification language should not distinguish two programs displaying the same behavior. On a theoretical level, this boils down to the requirement that a formula used for specification is invariant under bisimulation. So the Janin-Walukiewicz theorem basically says that the $\mu$-calculus is the “biggest” relevant specification language which a fragment of **MSO**.

It is also interesting to mention that the Janin-Walukiewicz theorem extends an important result of modal logic proved by Johan van Benthem [Ben76]: a first-order formula $\varphi$ is equivalent to a modal formula iff $\varphi$ is invariant under bisimulation. In the area of modal and temporal logics, the most common logics used as yardsticks (references against which the other logics are compared) are first-order logic and **MSO**. It follows from the Janin-Walukiewicz theorem and van Benthem characterization that the $\mu$-calculus is the counterpart of **MSO**, in the same way that modal logic is the counterpart of first-order logic. From a theoretical point of view, this makes the $\mu$-calculus an attractive extension of modal logic.

In order for a logic to be used as a specification language, it is important that there is a good balance between its expressive power and its complexity. By complexity, we usually refer to the complexities of the model checking problem and the satisfiability problem. The model checking problem was already mentioned before and consists in deciding whether a given formula holds on a given finite structure. The satisfiability problem consists in deciding whether for a given formula, there exists a structure in which the formula is true.

The model-checking problem for the $\mu$-calculus is $\text{NP} \cap \text{co-NP}$; the result can even be strengthened to $\text{UP} \cap \text{co-UP}$ [Jur98]. To obtain this upper bound, the idea is to use the connection between parity games and the $\mu$-calculus, which was observed by several authors, including E. Allen Emerson, Charanjit Jutla [EJ88].

\footnote{A non-deterministic Turing machine is unambiguous if for every input, there is at most one accepting computation. The complexity class UP (Unambiguous Non-deterministic Polynomial-time) is the class of languages problems solvable in polynomial time by an unambiguous non-deterministic Turing machine (for more details on this model of computation, see for instance [Pap94]).}
Chapter 1. Introduction

and Colin Stirling [Sti95]. Parity games play a fundamental role in game theory. A parity game is a game of which the winning condition is specified by a map assigning a bounded priority to each position of the board game. The winner of an infinite match depends on the priorities encountered infinitely often during the match.

It can be shown that the model checking problem for the $\mu$-calculus is equivalent to the problem of solving parity games, which consists in deciding which player has a winning strategy in a given parity game with an initial position. It was proved independently by Andrzej Mostowski [Mos91], E. Allen Ermerson and Charanjit Jutla [EJ91] that a winning strategy in a parity game may be assumed to be positional. That is, the move dictated by the strategy at a position of a match only depends on the actual position, and not on what has been played before reaching the position. This result implies that the complexity of solving a parity game is $\text{NP} \cap \text{co-NP}$. Later Marcin Jurdziński gave a tighter complexity bound, which is $\text{UP} \cap \text{co-UP}$ [Jur98]. It is an important open problem what is the exact complexity of solving parity games and in particular, whether this complexity is polynomial.

The satisfiability problem for the $\mu$-calculus is EXPTIME-complete [EJ88]. This was shown by E. Allen Ermerson and Charanjit Jutla, using automata theoretic methods. The basic idea of the automata theoretic approach is to associate with each formula an automaton that accepts exactly the structures in which the formula is true. It follows that solving the satisfiability problem for a formula is reduced to checking non-emptiness of an automaton. The non-emptiness problem for an automaton is to decide whether there exists a structure accepted by the automaton.

Similarly to the model checking problem, verifying whether there is a structure accepted by a given automaton is equivalent to checking whether a player has a winning strategy in an initialized infinite game associated with the automaton [NW96]. Furthermore, a winning strategy in the game would directly induce a structure accepted by the automaton.

The automata theoretic approach has also been useful for establishing other important results. For example, the Janin-Walukiewicz theorem mentioned earlier is proved using the correspondence between formulas and automata. In addition, David Janin and Igor Walukiewicz showed that there is a disjunctive normal form for the formulas of the $\mu$-calculus [JW95a]; a crucial part of the proof is based on the fact that we can determinize automata operating on infinite words.

The goal of these last few paragraphs was not only to give some insight about the complexity of the $\mu$-calculus, but also to illustrate how the theory of the $\mu$-calculus benefits from the connections between different formalisms, such as game theory, automata theory and, obviously, logic. This feature is not specific to the $\mu$-calculus: the same holds for all temporal logics and on a broader scale, this is a phenomenon that is characteristic of many areas of mathematics. Nevertheless,
it is still a very enjoyable aspect of the \( \mu \)-calculus.

Now, from what we have seen, the \( \mu \)-calculus seems a well-suited specification language, as it combines a great expressive power and manageable decision procedures. But there is a drawback: the \( \mu \)-calculus is probably not the most understandable way to specify behaviors. Most people would have a difficult time understanding the meaning of a formula of the \( \mu \)-calculus with alternation depth greater than 2. In that respect, other temporal logics, such as LTL, CTL and CTL*, are more convenient.

That being said, it is still the case that: the \( \mu \)-calculus provides a uniform framework containing all specification languages; it is characterized by a rich and interesting mathematical theory; despite its difficult interaction with human thinking, it has direct practical applications in the area of specification. For these reasons, the \( \mu \)-calculus has become a significant formalism in the landscape of modal logic and specification.

In this thesis, we consider some important theoretical aspects of the \( \mu \)-calculus, namely axiomatizability, expressivity, decidability and complexity. One running topic through the thesis is exploring the \( \mu \)-calculus through its “fine-structure”. Or to put it differently, we investigate the \( \mu \)-calculus by focussing on restricted class of models, special fragments of the language, etc. This approach is motivated by the fact that the \( \mu \)-calculus is a complex and powerful system.

In Chapters 3 and 4, we restrict our attention to special classes of models, namely trees. Trees are particularly relevant structures for any logic that is invariant under bisimulation. Such logics have the tree property; that is, a formula is satisfiable iff it is satisfiable in a tree. In Chapter 3, we consider the question of the axiomatization of the \( \mu \)-calculus. This problem is notorious for its difficulty, but it turns out that when focussing on finite trees, the proof of the completeness of the axiomatization becomes much simpler. In Chapter 4, we deal with the question of the expressive power of the \( \mu \)-calculus in the context of frames (which are transition systems without any labeling). Again we investigate this question in the restricted setting of trees.

In Chapter 5, instead of having restrictions on the structures, we consider some special fragments of the language. The main contribution of that chapter concerns a characterization of what we call the continuous fragment. As we will see, the continuous fragment is a good candidate for approximating the “computational part” of the \( \mu \)-calculus. Chapter 6 is slightly different than the other chapters, as it concerns the formalism XPath [BK08]. The goal of that chapter is to show how results in the area of modal logic can help for the understanding of XPath. One of these results was shown in Chapter 5.

The last chapter is also concerned with special classes of models, but the perspective with respect to the “fine-structure” approach is in effect reversed. Instead of looking at specific classes of models, we consider more general structures, namely coalgebras. Coalgebras are an abstract version of evolving systems and generalize the notion of labelled transition systems or Kripke models. In
Chapter 7, we extend the automata theoretic approach for the $\mu$-calculus to the setting of coalgebras.

We give now a more detailed overview of the content of each chapter.

Axiomatizability

Chapter 3 In the same paper where he introduced the $\mu$-calculus [Koz83], Dexter Kozen also suggested an axiomatization. The completeness of that axiomatization remains an open problem for many years. Eventually Igor Walukiewicz [Wal95] provided a proof which is based on automata theory, game theory and classical logic tools such as tableaux. The proof is also well-known for its difficulty.

In Chapter 3, we propose an easier proof in the restricted setting of the $\mu$-calculus on finite trees. On finite trees the expressive power of the $\mu$-calculus is rather limited: any formula of the $\mu$-calculus is equivalent to a formula of alternation depth 1. Nevertheless, the completeness proof we provide is not a simplification of the original proof given by Igor Walukiewicz. The technique we use consists in combining an Henkin-type semantics for the $\mu$-calculus together with model theoretic methods (inspired by the work of Kees Doets [Doe89]).

We hope that this different approach towards completeness might contribute modestly to a better understanding of the problem. This method might also help to prove other completeness results and we give two examples in the chapter. The first one concerns a complete axiomatization of the graded $\mu$-calculus on finite trees. The other example applies to extensions of the $\mu$-calculus with shallow axioms [Cat05] on finite trees.

This chapter is based on the paper “An easy completeness proof for the $\mu$-calculus on finite trees”, co-authored by Balder ten Cate and published in the proceedings of FOSSACS 2010.

Expressive power

Chapter 4 The Janin-Walukiewicz theorem concerns the expressive power of the $\mu$-calculus on the level of models, i.e. transition systems equipped with a valuation (stating which atomic propositions are true at each node). In Chapter 4, we shift to the context of frames, which are transition systems without any valuation. The truth of a formula in a frame involves a second-order quantification over all possible valuations.

As opposed to the case of modal logic, very little is known about the expressive power of the $\mu$-calculus on frames. This chapter compares the expressive power of the $\mu$-calculus and MSO on frames, in the particular case when the frames have a tree structure. More specifically, we provide a characterization of those MSO formulas that are equivalent on trees (seen as frames) to a formula of the
\(\mu\)-calculus. This characterization is formulated in terms of natural structural criteria, namely closure under subtrees and \(p\)-omorphic images. The result might be compared to the Janin-Walukiewicz theorem, the main differences being the context (frames vs. models), and our more restricted setting (trees vs. arbitrary models).

This chapter is based on the paper “Frame definability for classes of trees in the \(\mu\)-calculus” co-authored by Thomas Place and published in the proceedings of MFCS 2010.

**Chapter 5** We present syntactic characterizations of semantic properties of the \(\mu\)-calculus, the two main ones being the continuous fragment and completely additive formulas. A formula \(\varphi\) of the \(\mu\)-calculus is continuous in a proposition letter \(p\) if the truth of \(\varphi\) at a given node only depends on the existence of finitely many points making \(p\) true. The name “continuity” originates from the direct connection between this fragment and the notion of Scott continuity, widely used in theoretical computer science. One of the most interesting features of a continuous formula is that its least fixpoint can be constructed in at most \(\omega\) steps.

The completely additive fragment corresponds to distributivity over countable unions, which, in the case of the \(\mu\)-calculus, was studied by Marco Hollenberg [Hol98b]. Using a characterization of this fragment, Marco Hollenberg obtained an extension of the Janin-Walukiewicz theorem for \(\mu\)-programs [Hol98b] (which is what motivated the study of the completely additive fragment). Inspired by our results for the continuous fragment, we propose an alternative proof for the characterization of the completely additive fragment. Unlike the original argument, this proof provides a direct translation from the completely additive fragment to the adequate syntactic fragment.

This chapter is based on the paper “Continuous fragment of the \(\mu\)-calculus” published in the proceedings of CSL 2008 and on a submitted paper “Syntactic characterizations of semantic fragments of the \(\mu\)-calculus” co-authored by Yde Venema.

**Chapter 6** This chapter is concerned with the expressive power of a fragment of CoreXPath. XPath is a navigation language for XML documents and CoreXPath has been introduced to capture the logical core of XPath [GKP05]. The basic idea of the chapter is to exploit the tight link between CoreXPath and modal logic. CoreXPath is essentially a modal logic evaluated on specific models (which are finite trees with two basic modalities). The main difference between modal logic and CoreXPath is that the syntax for XPath is two-sorted: it contains both formulas (which corresponds to subsets of the model) and programs (which corresponds to binary relations).

In this chapter, we combine well-known results of the \(\mu\)-calculus in order to obtain results about the expressive power of CoreXPath. One of the results that
we use is the adaptation of the Janin-Walukiewicz theorem for $\mu$-programs. This result was (re-)proved in the previous chapter.

This part of the thesis is based on the paper “Modal aspects of XPath” co-authored by Balder ten Cate and Tadeusz Litak and which is an invited paper for M4M 2007.

**Decidability and complexity**

**Chapter 7** In this chapter, we extend the notion of automaton to the setting of coalgebras. The aim of the theory of coalgebras is to provide a uniform framework to describe evolving systems, Kripke models being a key example. It is then not surprising that the definition of coalgebraic logic was inspired by modal logic. Roughly, there are two kinds of coalgebraic logic: one using nabla ($\nabla$) operators [Mos99], the other being based on the notion of predicate lifting [Pat03]. Similarly to what happens in modal logic, we can extend coalgebraic logic with fixpoint operators and obtain a coalgebraic $\mu$-calculus.

As mentioned earlier, the automata theoretic approach has been very successful for the $\mu$-calculus. Automata for the coalgebraic $\mu$-calculus using nabla operators have been introduced by Yde Venema [Ven06b]. The goal of this chapter is to contribute to the development of the automata theoretic approach for coalgebraic $\mu$-calculus based on predicate liftings. More specifically, we introduce the notion of an automaton associated with a set of predicate liftings. We use these automata to prove the decidability of the satisfiability problem and obtain a small model property. We also obtain a double exponential bound on the complexity of the satisfiability problem.

This chapter is based on the paper “Automata for coalgebras: an approach via predicate liftings” co-authored by Raul Leal and Yde Venema and published in the proceedings of ICALP 2010.

We mentioned earlier that there exist connections between the $\mu$-calculus and other formalisms, the two major ones being game theory and automata theory. We can think of the exploitation of these connections as being methods for approaching the $\mu$-calculus. From that point of view, the five chapters that we described can be seen, independently from the content, as a playground for these methods: how they interact and what they can be used for.

In the following chapters, we often make use of the links with automata theory and game theory. These two theories are themselves deeply connected to each other. One of the main reasons (in our context) is that the terminology of game theory is particularly adequate to describe the run of an automaton on branching structures. The fact that a tree is accepted by an automaton is usually reduced to the existence of a winning strategy for a player in a game associated with the automaton.
The most obvious place in the thesis where logic, automata and games are intertwined is Chapter 7. This chapter can be seen as an illustration of the efficiency of the automata theoretic approach. Automata are in effect an alternative way of thinking about formulas. One of their advantages is that they capture the algorithmic aspect of the $\mu$-calculus, while not having the logical complexity resulting from an inductive definition (unlike formulas). Game theory also comes into play in this chapter: it offers a nice framework to interpret automata and formulate proofs.

Even though there is no explicit mention of automata in the Chapters 4 and 5, we could still think of these chapters as using the connections between automata, games and logic. A useful result in both chapters is the equivalence between the model checking problem for a formula and solving a certain parity game called the evaluation game. The evaluation game is the acceptance game of the alternating $\mu$-automaton associated with the formula.

Now we do not only use game theory and automata theory as formalisms to represent formulas in a more intuitive or convenient way, but also as reservoirs of available results. For example, proofs in Chapter 7 rely on the fact that a strategy in a regular game may be assumed to be a finite memory strategy. Interestingly enough, a classical automata result (the determinization of automata on infinite words) plays an important role in the proof of this fact. Another (less direct) example is given in Chapters 4 and 5. The proofs in these chapters use in an essential way the existence of a disjunctive normal form for fixpoint logics. As mentioned earlier, a key ingredient for the proof of this last result is of automata theoretic nature (and is again the determinization of automata on infinite words).

Finally, to the existing arsenal, we add a new method in Chapter 3. As explained in the overview of this chapter, this method is inspired by model theory. The idea is to introduce a notion of rank which plays the same role in our proof as the notion of quantifier depth in model theory. Using this notion, we can transfer model theoretic arguments that work by induction on the alternation depth to the setting of the $\mu$-calculus. In our case, the model theoretic argument originates from Kees Doets’ work [Doe89].