BOUNDED MODEL-CHECKING FOR BRANCHING-TIME LOGIC

Rotem Oshman

BOUNDED MODEL-CHECKING FOR BRANCHING-TIME LOGIC

Research Thesis

Submitted in partial fulfillment of the requirements for the degree of
Master of Computer Science

Rotem Oshman

Submitted to the Senate of the Technion - Israel Institute of Technology

SIVAN , 5768 Haifa June, 2008

The research thesis was done under the supervision of Prof. Orna Grumberg in the Faculty of Computer Science.

The generous financial help of the Technion is gratefully acknowledged.

Contents

List of Figures

List of Tables

Abstract

The model-checking problem asks, given a model of a design and a specification in temporal logic, whether the model satisfies the specification. Model-checking industrial designs is a computationally difficult problem, because the state-space that must be explored during model-checking grows exponentially with the number of variables. One way to handle large designs is bounded model-checking (BMC), in which we search for a bug or counter-example of bounded size k with regard to the specification. The existence of such a bug is encoded as a Boolean formula, which is satisfiable if and only if there is a bug in the model; a SAT solver is then used to check if the formula is satisfiable or not. If the formula is satisfiable, there is a bug in the design, and a satisfying assignment is returned to the user as a trace. If the formula is unsatisfiable, we cannot in general conclude that there is no bug; we must increase the bound, and search for "larger bugs".

Bounded model-checking is usually applied only to linear-time properties, which are formulas that can only talk about each computation path separately. In contrast, branching-time properties refer to the computation tree of the program. BMC for branching-time properties is more difficult than BMC for linear-time properties, because it is not known in advance what shape a counter-example to the property will have. Previous works on BMC for branching-time logic have based their BMC encodings on worst-case syntactic analysis of the formula. This approach leads to inefficient encodings, which use a number of variables that grows exponentially as the formula becomes more complex. It also leads to formulas which have satisfying assignments that do not provide enough information to construct a meaningful trace to return to the user.

In this dissertation we suggest a new way to perform BMC for branching-time specifications. In our approach, we do not constrain the shape of the counter-example in advance: we let the SAT solver choose both the states and the edges of the counter-example. The resulting encoding is compact, grows polynomially with both the bound and the complexity of the formula, and returns minimal counter-examples. A satisfying assignment to the formula encoded also includes annotation, which allows us to explain to the user *why* it represents a counter-example to the specification. We present an encoding for the universal fragment of μ-calculus, and then derive a specialized encoding for the alternation-free fragment.

In addition to finding bugs, BMC can sometimes be used to prove properties. A *com-*

pleteness threshold is an upper bound, such that if we have not found a bug when the bound reaches the completeness threshold, then there is no bug. We develop a completeness threshold for ACTL properties. We also develop a *dynamic completeness criterion*, which is a Boolean formula that tells us whether there is any hope for finding a counter-example with a larger bound than the current bound. If the dynamic completeness criterion is unsatisfiable, we know that no counter-example will be found with a larger bound, and we can conclude that the specification is satisfied. If the dynamic completeness criterion is unsatisfiable, we must increase the bound and continue searching.

List of Symbols and Abbreviations

Abbreviations

BMC	bounded model-checking
SAT	the satisfiability problem
ACTL, ECTL	the universal and existential fragments of CTL (respectively)

Symbols and Operators

$\models$	satisfaction relation between Kripke structures and formulas or automata
$\lhd_q$, $\circledcirc_q$	progress relations on rank vectors
2^A	the powerset of the set A
L_μ	the modal μ-calculus
$\Box L_\mu$, $\Diamond L_\mu$	the universal and existential fragments of the modal μ-calculus (respectively)
$[\![\varphi]\!]^e_M$	denotation of φ in M w.r.t. environment e
domain(Π)	the states that participate in the proof Π

Variables

Kripke Structures, Formulas and Automata

M	Kripke structure or model
S	model states
s_0	initial state
R	transition relation
L	labeling function
AP	atomic propositions
φ, ψ	temporal formulas
p	atomic proposition
A	automaton
Q	automaton states
δ	automaton transition function

F	Büchi automaton acceptance condition
Ω	parity automaton priority function
$<_Q$	partial order on accepting and rejecting sets in a weak automaton

Temporal Proofs

Π	Namjoshi-style temporal proof
Υ	partial proof
I_q	proof invariant for automaton state q
$\rho_q(s)$	rank vector assigned by automaton state q to model state s
$\sigma_q(s)$	integer rank assigned by automaton state q to model state s
PS	proof-successor assignment
$PS_q(s)$	proof-successor assigned by automaton state q to model state s in PS

Boolean Variables, Predicates and Formulas

u_i	i-th state in the counter-example
$x_i^q, x_i^{q,t}$	indicator bit for whether u_i belongs to I_q
ρ_i^q	rank assigned by q to u_i
R	Boolean representation of the transition relation
I	Boolean representation of the initial state predicate
P_q	Boolean representation of the progress relation for q
$\mathsf{PRF}_{M,A,k}$	Boolean formula searching for a counter-example of size k
$\mathsf{PRF}^{\mathsf{sym}}_{M,A,k}$	same as $\mathsf{PRF}_{M,A,k}$ but with symmetry-breaking
$\mathsf{PRF}^{\mathsf{exp}}_{M,A,k}$	same as $\mathsf{PRF}_{M,A,k}$ but with successors encoded explicitly
$\mathsf{CMP}_{M,A,k}$	Boolean formula for the dynamic completeness criterion

Chapter 1

Introduction

The model-checking problem deals with verifying whether a given model conforms to a specification. Model-checking has found many practical uses in the industry, particularly in the verification of hardware designs. Model-checking large designs can be computationally difficult because of the *state explosion problem*: as the number of variables or gates in the design increases, the number of states that need to be explored during model-checking grows exponentially.

One way to alleviate the problem is through use of OBDD-based algorithms. OBDDs are a generally compact way to represent sets of states, and Boolean operations such as disjunction (set union) and conjunction (set intersection) can be implemented efficiently on OBDDs. Model-checkers that use OBDDs are called *symbolic model-checkers*, as they manipulate sets of states rather than individual states during their execution. However, even symbolic model-checkers are often unable to deal with the size and complexity of modern industrial designs.

Bounded model-checking (BMC) is another approach to combatting the state-explosion problem. In BMC, instead of computing the set of states in the design that satisfy the specification, one uses a SAT solver to try to find a bug — a trace or behavior of the program that does not satisfy the specification. The Boolean formula passed to the SAT solver in BMC is generally composed of two components: first, constraints that encode the existence of a trace or behavior of some bounded size in the model; and second, constraints that force the behavior to satisfy the *negation* of the specification. A satisfying assignment to the formula then represents a bug in the design.

In bounded model-checking we only search for bugs of *bounded* size. A typical BMC algorithm starts with some small bound, and increases it in each iteration until a bug is found, or until the bound reaches some threshold which allows us to conclude that no bug exists. Such a threshold is called a *completeness threshold*.

BMC has gained popularity in the industry, because it allows one to find bugs in very large designs. However, attention has mostly been focused on BMC for *linear-time* speci-

fications, which are specifications that refer to all computation paths of the program, but only consider each path separately. In constrast, *branching-time* specifications refer to the computation tree of the program. Branching-time specifications can express conditions such as the existence of a "good way to continue" from some point in the computation tree, or talk about "all possible futures" from some point in the execution.

All linear-time specifications admit counter-examples in the form of a finite or infinite path (a trace). Therefore, in BMC for linear-time logic, one encodes a path of bounded length, and constrains it to represent a counter-example. BMC for branching-time logic is a somewhat thornier problem, because it is not known in advance what *shape* the counter-example will take; the counter-example can be a path, a tree, or in general any kind of graph. In fact, for some branching-time specifications there is no natural notion of a counter-example: for example, to disprove a specification such as "there exists a path that reaches some good state", one needs to show that all paths in the model do not reach a good state — in other words, the counter-example needs to include all paths of the model. Here we restrict attention to *universal* branching-time properties, which can only talk about all possible futures, not the existence of a future. Universal properties have tractable counter-examples. However, even with this restriction, it is still not clear in advance what shape the counter-example to a given property will take.

Previous works on BMC for branching-time logic have taken a "worst-case analysis" approach: based on syntactic analysis of the formula, one essentially encodes the largest and most complicated counter-example that may be required to disprove the formula, limited by the bound, which corresponds to the depth of the counter-example. This pessimistic approach leads to large counter-examples. In the worst-case, the number of variables needed to represent the counter-example is exponential in the bound. In addition, if a bug is found, the assignment returned by the SAT solver is not very informative: as a result of the worst-case analysis performed on the formula, the assignment may describe parts of the model that are not needed to disprove the formula, and it provides no way to distinguish between useful and useless parts.

In this work we suggest a new approach to BMC for branching-time logic. Instead of constraining the shape of the counter-example in advance as was done in previous work, we let the SAT solver choose both states and edges of the counter-example. We add *local* constraints, to ensure that the counter-example satisfies the negation of the formula. Each local constraint on a model state can either constrain the state itself, or it can require the existence of an edge to another state that satisfies certain criteria; local constraints do not make any requirements beyond the immediate successors of a state.

Our encoding is *compact* in the sense that no model state is encoded more than once, and each state can be "re-used" to justify many different subgoals as part of disproving the entire formula. As a result, the counter-example returned in our approach contains the minimal

number of states a counter-example can have. We also return an annotation explaining which state was used to justify which subgoals.

We show a static completeness threshold for the universal branching-time logic ACTL, based on syntactic analysis of the formula and the recurrence diameter of the model. In addition, following the work of Wang in [42], we suggest a *dynamic completeness criterion*, which can be used to halt the BMC when it becomes clear that increasing the bound further will not lead to the discovery of a counter-example. The dynamic completeness criterion is a Boolean formula whose unsatisfiability indicates that there is no structure that may be extended into a counter-example when the bound is increased. We point out a mistake in [42] which leads to unsoundness of the completeness criterion presented there.

Finally, we present experimental results, which show that for complicated branching-time specifications, our approach performs better than the recent encoding of [42]. Our encoding is able to falsify more formulas, and also returns counter-examples that are smaller by orders of magnitude.

1.1 Related Work

1.1.1 Bounded Model-Checking

In bounded model-checking (BMC) [4], the problem of finding a counter-example of some bounded size for a given formula is reduced to satisfiability of Boolean formulas. The motivation behind the reduction is to leverage recent advances in SAT solver technology in order to solve large and complex verification problems. The main idea is to encode the existence of a bounded counter-example in the form of a Boolean formula, such that any satisfying assignment to the formula represents a counter-example. A SAT solver is then used to determine whether or not there exists a satisfying assignment.

The specification language used in [4] is LTL, and many subsequent works dealt with LTL or various other linear-time specification mechanisms; e.g., [17], [28] and [21] for LTL, [19] for weak alternating Büchi automata, [23] for linear-time μ-calculus, [18] for LTL with past (PLTL) and [37] for timed automata, to name but a few.

When dealing with linear-time properties, which characterize the desired behavior of all computation paths of the program, a counter-example to the property is always a finite or infinite path that does not comform to the characterization. If the path is finite, the bound k in bounded model-checking corresponds to the length of the path; a path of length k is referred to as a *k-path*. Among infinite paths we are only interested in "lasso-shaped" paths, which enter an infinite loop after some point. It can be shown that any LTL formula which has a counter-example in the form of an infinite path also has a lasso-shaped counter-example. A lasso-shaped path with a prefix of length k before it starts repeating is called a *k-loop*.

In [4], the translation to a Boolean formula relies on *bounded semantics for LTL*. The standard (unbounded) semantics for LTL define when a formula is satisfied by an infinite path; the bounded semantics define satisfaction over k-paths and k-loops. The bounded semantics is sound with regard to the unbounded semantics: if a formula is satisfied with regard to the bounded semantics by a finite prefix of a path, then the formula is satisfied with regard to the unbounded semantics by the entire (infinite) path; for a loop, satisfaction in the bounded semantics is equivalent to satisfaction in the unbounded semantics. The bounded semantics is also complete, in the sense that if a formula is satisfied with regard to the unbounded semantics by some infinite path π in a model, then there exists a (possibly different) path π' and a bound k, such that the formula is either satisfied by the finite prefix of length k of π', or π' is a k-loop that satisfies the formula.

The existence of a k-path or k-loop that represents a counter-example to a given formula with regard to the bounded semantics can be formulated as a Boolean formula over k states. The k states are constrained to represent either a k-path or a k-loop by unfolding the transition relation of the model $k-1$ times and constraining the first state to be an initial

state. (For a k-loop, additional constraints are necessary to require a back-edge.) In addition, constraints are added to ensure that the k-path or k-loop satisfies the *negation* of the original formula with regard to the bounded semantics. A SAT solver is used to check for the existence of a satisfying assignment, which represents a counter-example to the original formula. If a satisfying assignment exists, it is concluded that the model does not satisfy the formula, and the assignment is returned to the user as a bug trace. If, however, the formula is unsatisfiable, it cannot usually be concluded that the model satisfies the formula. Instead, one must either increase the bound k and try again, or try to determine that the formula is satisfied by using a completeness criterion.

1.1.2 Completeness Criteria

Bounded model-checking is typically used to find counter-examples of some bounded size. If a counter-example of size k cannot be found, it is generally not possible to conclude that there is no counter-example. However, there are circumstances when one might safely do so.

A *completeness threshold* is a predetermined bound, such that if the formula has a counter-example at all, it must also have a counter-example whose size is smaller than the completeness threshold. The completeness threshold typically depends on both the formula and the model. A completeness threshold can be used to verify properties using bounded model-checking: if the bound has reached the threshold and no counter-example has been found, it is safe to conclude that no counter-example exists and the model satisfies the formula.

The original completeness threshold presented in [4] is very pessimistic, although a tighter bound is presented for the class of lasso-shaped Kripke structures. Tighter completeness thresholds are developed for various classes of temporal properties in [8], through the use of automata-theoretic techniques, which will be discussed in Section 1.1.3. Completeness thresholds in general, and the thresholds of [8] in particular, rely on measures such as the *diameter* of the model, which is the longest shortest path between any two reachable states in the model, and the *recurrence diameter* of the model, which is the length of the longest simple (loop-free) path between any two reachable states.

The diameter and recurrence diameter are typically difficult to compute, making it impractical to compute a completeness threshold in many cases. In [4] it is shown how to test if the diameter (resp. recurrence diameter) is larger than a given integer k by constructing a Boolean formula that is satisfiable iff the diameter (resp. recurrence diameter) is larger than k, and testing its satisfiability. The diameter and recurrence diameter can be computed by testing successively greater values of k in this manner until the formula becomes unsatisfiable, and then we know that the diameter has been reached. However, if the diameter or recurrence diameter is large, the Boolean formula may become intractable for the SAT solver due to its large size. Several methods for simplifying the computation of the recurrence

diameter are discussed in [25], but it is still a difficult problem.

Another way to verify safety formulas using SAT is *temporal induction*, first suggested in the context of bounded model-checking in [35]. A property p is said to be *k-inductive* if for every path of length $k+1$, if the first k states satisfy p then the last state satisfies p also. To prove that a property is k-inductive, one tests the satisfiability of the Boolean formula that encodes the existence of a path of length $k+1$, where the first k states satisfy p but the last state does *not* satisfy p. If this formula is unsatisfiable, then p is k-inductive. If, in addition, every path of length k starting from the initial state contains only states that satisfy p — which again is easy to check using a SAT-solver — we can conclude that all the reachable states satisfy p.

If the proof rule outlined above fails, one either increases k or strengthens the property p (referred to as the inductive invariant). Induction is theoretically complete: if the property $p \wedge r$ is used as an invariant, where r expresses reachability, then a 1-induction proof succeeds in every system where all reachable states satisfy p. However, computing the reachability condition r is usually not practical, so better ways are needed to ensure termination. One way is suggested in [2], where every time the proof fails, the inductive invariant is strengthened to rule out the unreachable states that prevented the proof from succeeding.

1.1.3 The Automata-Theoretic Approach

The automata-theoretic approach to model-checking, championed by Vardi, Wolper and Kupferman, advocates the use of automata to express specifications. The main ideas are summarized in [5].

Many of the results in the automata-theoretic approach to linear-time logic are presented in [41]. Linear-time logic specifications correspond to automata on finite or infinite words; in particular, LTL specifications can be translated to nondeterministic Büchi automata on infinite words. Questions such as validity and model-checking can be translated to automata-theoretic problems such as automata emptiness, which asks whether a given automaton accepts any word, and language inclusion, which asks whether all the words accepted by one automaton are also accepted by a second automaton.

The automata-theoretic approach to model-checking has proven useful in bounded model-checking. In [8], the automata-theoretic approach is used to derive an efficient translation into SAT of the bounded model-checking problem. The idea is that, given a model M and an LTL property $\varphi = Af$, one constructs the automaton $A_{\neg\varphi}$ corresponding to the *negation* of f. If this automaton accepts any computation path of M, then $M \not\models \varphi$. To test whether any computation path is accepted, the cross-product $M \times A_{\neg\varphi}$ is constructed, and bounded model-checking is used to find an accepting run of $M \times A_{\neg\varphi}$. If such a run exists, it represents a computation path of M that is accepted by $A_{\neg\varphi}$, and therefore $M \not\models \varphi$. In [8] this is referred to as *the semantic translation*; it yields smaller formulas, with fewer variables,

than the formulas produced by the translation of [4], which is referred to as *the syntactic translation*. In addition, techniques such as automata minimization can be employed, and the resulting encoding is also simpler and more uniform with regard to the specification language. The automata-theoretic approach also yields completeness thresholds for all LTL properties [8].

Another automata-theoretic translation for linear-time logic BMC is presented in [19]. In this work the specification mechanism used is weak alternating Büchi automata on infinite words. Alternating automata are automata whose transition relation can combine existential transitions, which specify the set of states to which it is possible to transition, and universal transitions, which specify a set of states to which the automaton *must* transition. Such automata are exactly as expressive as nondeterministic automata, but they can be exponentially smaller. *Weak* automata have a particular structure that can be exploited to simplify model-checking [26]. In [19], the authors borrow ideas from the world of symbolic model-checking to derive an efficient encoding for weak alternating Büchi automata. Our work, although developed independently for the most part, is similar to theirs in many respects. The notion of simulating the run of a symbolic model-checker in the Boolean formula (Section 3.3) came from [19].

For branching-time logic, the corresponding automata-theoretic formalism is automata on trees. The automata-theoretic approach to model-checking for branching-time logic is described in [27]. CTL formulas can be translated into alternating automata on infinite trees with a Büchi acceptance condition. However, to express general μ-calculus properties, a more expressive acceptance condition is necessary. The *parity acceptance condition* for tree automata was suggested by Mostowsky in [31]. In [13], Emerson and Jutla showed that μ-calculus formulas can be expressed as alternating tree automata with a parity acceptance condition. Alternating parity tree automata are presented in [43] in a form that is the closest to the form we will use in this work, and the translation from μ-calculus to automata and vice-versa is also described in [43] in a very natural way.

1.1.4 Counter-Examples and Proofs

One of the chief concepts around which bounded model-checking revolves is the notion of a *counter-example*, which is generally a sub-structure of the model that is sufficient to demonstrate that the model does not satisfy a temporal property. As already discussed in Section 1.1.1, a counter-example for a linear-time property is a path in the model. However, for branching-time properties it is less clear what form a counter-example should take. For example, the CTL property EFp says that there exists a path along which we eventually reach a state labeled with p. To convince a user that the property does not hold in a model, we must be able to show that along *all* paths we never reach a state labeled with p. To do so, the counter-example must contain the entire reachable fragment of the model. Even when

dealing only with universal path quantifiers in the property — which means the negation contains only existential path quantifiers — it is still not clear just what a counter-example should be. In [10] it is shown that universal CTL (ACTL) formulas always admit *tree-like counter-examples*, which are structures in which the strongly-connected components (SCCs) form a finite tree, and each SCC is a cycle.

Counter-examples are generally restricted to the case where the property being checked does not hold in the model, and they are also restricted in the most part to logics that only allow universal path quantifiers (e.g., LTL and ACTL). To witness that a model does satisfy a property we can use a *proof* or certificate. Many different proof systems for various temporal logics have been suggested; the one that will be used in this work is Namjoshi's proof system for μ-calculus [32], which will be explained in detail in Section 2.4. The feature that makes it useful for our purposes is that its conditions are local: to verify that a proof is valid, one need only check a series of *local* conditions, which refer at most to a state's immediate successors in the model. Other proof systems, such as Stirling's proof rules [38], keep track of states visited along the current proof branch; in Namjoshi's system such "book-keeping" is not required, and ranks are used instead. Relying on the proof rules from [38] yields a fundamentally different bounded model-checking algorithm from the one we present here: this is the encoding presented in [42]. The differences will be discussed at length in Chapter 5.

Namjoshi's proof system is based on the close connections between the μ-calculus, parity automata and parity games ([14], [15]). A Namjoshi-style proof for $M \models A$, where M is a model and A is an automaton, can be thought of as representing a winning strategy in the parity game corresponding to M and A. The proof system is described in Section 2.4.

Proofs and counter-examples are closely related concepts. A counter-example witnesses that the model does not satisfy the formula, so it can be seen as a proof for the *negation* of the formula. A proof can also be used as a counter-example when it is not possible to present the offending behavior as a simple sub-structure of the model, as in the case of EFp. Another advantage to using proofs over counter-examples in this case is that a proof can summarize many bad behaviors [32], whereas a counter-example typically represents only one behavior. This can be useful for applications such as counter-example guided abstraction-refinement (CEGAR), where information about spurious bad behaviors is used to refine the abstraction. Proofs are also useful as *annotation* for counter-examples ([36], [6]): they can help explain why the counter-example causes the formula to fail. And finally, as discussed in [32], a proof can be exponentially more succint than a counter-example.

The main idea of our work, as in [42], is to use a SAT solver to find a counter-example in the form of a proof. The *domain* of the proof is the set of model states that participate in the proof; this is used as an actual counter-example, because it represents a sub-structure of the model that is sufficient to exhibit the bad behavior that causes the formula to fail. In

addition, we obtain an annotation, which tells us which states of the counter-example satisfy which subformulas in the negation of the original formula.

1.1.5 Bounded Model-Checking for Branching-Time Logic

The bounded model-checking problem for branching-time logic has received less attention than bounded model-checking for linear-time logic. To the best of our knowledge, three different approaches have emerged. They are discussed in detail in Chapter 5. Here we give a brief overview.

In [33], bounded model-checking for ACTL is achieved by encoding the counter-example as a collection of bounded paths. This work uses similar ideas to [4]. The authors develop a bounded semantics for ACTL, similar to the way bounded semantics for LTL are developed in [4], and based on the bounded semantics they constrain the paths encoded to represent a counter-example to the property. The number of paths that need to be encoded in order to falsify a given formula is exponential in the number of temporal operators, in the worst case. The number of variables in the resulting encoding is exponential in the formula but polynomial in the bound. The counter-example obtained is also not very informative — it is a collection of paths of the same length, with no annotation to indicate which parts of which paths contributed to the offending behavior. The approach of [33] is extended to ACTL*, the universal fragment of CTL*, in [45], using the same ideas.

An approach more similar to our own is taken in [42], where a counter-example in the form of a proof is sought. The proof system used in [42] is Stirling's proof system from [38]. A Stirling-style proof is a tree, with constraints on the successors to each node in the tree. In branches corresponding to greatest-fixpoint subformulas, the leaves are required to contain a model state that has previously appeared in the branch; in a branch corresponding to a least-fixpoint subformula the model states that appear along the branch are required to be distinct from each other. Consequently, the proof obligations are not entirely local; e.g., to ensure that a state does not appear in a branch, one has to refer to all the states that appear in the branch. In [38] this non-locality is hidden away by having the tree nodes "remember" which states appear in their branch, but the essence is still the same. The encoding that is developed in [42] is exponential in the bound in the worst case, although for ACTL properties it is polynomial in the bound and exponential in the formula. In addition, the counter-example is not informative.

The third approach is our own approach, which uses Namjoshi's proof system from [32] as a basis. We are able to obtain a very compact encoding, which is polynomial in both the bound and the formula, and yields informative counter-examples.

All three approaches offer accompanying completeness criteria, and all are based on the same idea: to dynamically test whether or not the bound should be increased, we construct a relaxed Boolean formula, which allows some "open ends". For example, in a witness for

the formula EGp, which must typically be a lasso-shaped path, we will allow the loop not to close, and accept a finite non-looping path whose states all satisfy p. In a formula like $E[p\,U\,q]$ we might accept in the relaxed version a path whose states all satisfy p, and not require that one of the states satisfy q. The idea is that the relaxed formula represents a *beginning* for a counter-example, which might be extended into a full counter-example by adding more states. If the relaxed version is unsatisfiable, then there is no hope for finding a counter-example, and we can terminate at the current bound and conclude that the model satisfies the formula. The completeness criterion for [33] is presented in [46], and the completeness criterion for [42] is presented in the original paper. However, we discovered that the completeness criterion of [42] is unsound: it is too strict, and may cause the model-checker to terminate and conclude that the formula is satisfied when in fact increasing the bound will lead to the discovery of a counter-example. The unsoundness is easily remedied by removing some constraints from the completeness criterion.

Chapter 2

Preliminaries

2.1 Kripke Structures

To represent finite-state programs, we use *Kripke structures*. A Kripke structure is a tuple $M = (S, s_0, R, L)$, where S is the set of states, s_0 is the initial state, $R \subseteq S \times S$ is a transition relation and $L : S \to 2^{AP}$ is a labeling function.

For the purpose of reducing the model-checking problem to Boolean satisfiability, we will assume the following Boolean representation for Kripke structures.

- The state-space S is represented as the set of binary vectors of length n, $S = \{0,1\}^n$, where n is the number of Boolean variables or gates in the program or circuit;
- The initial state s_0 is identified by a Boolean predicate $\mathsf{I} : S \to \{0,1\}$, such that for all $s \in S$, $\mathsf{I}(s) = 1$ iff $s = s_0$;
- The transition relation R is represented by a Boolean predicate $\mathsf{R} : S \times S \to \{0,1\}$, such that for all $s, t \in S$, $\mathsf{R}(s,t) = 1$ iff $(s,t) \in R$;
- The labeling function L is represented by a set $\{\mathsf{L}_p \mid p \in AP\}$ of Boolean predicates, such that for all $p \in AP$ and $s \in S$, $\mathsf{L}_p(s) = 1$ iff $p \in L(s)$.

We say that a model $N = (S', s'_0, R', L')$ is a *submodel* of $M = (S, s_0, R, L)$ if $S' \subseteq S$, $s'_0 \in S'$ (we do not require that $s'_0 = s_0$), $R' = R|_{S'} = R \cap (S' \times S')$, and $L'(s) = L(s)$ for all $s \in S'$.

2.2 The μ-Calculus

The modal μ-calculus L_μ [24] is an expressive branching-time logic. Many widely-used temporal logics, including LTL, CTL and CTL*, can be translated into μ-calculus, rendering it a convenient low-level property specification language.

2.2.1 The syntax of μ-calculus

Assume a set AP of atomic propositions and a set Var of fixpoint variables. The formulas of μ-calculus, in negation normal form (NNF), are given by the following grammar:

$$\varphi \quad ::= \quad p \mid \neg p \mid X \mid \Diamond\psi \mid \Box\psi \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \mid \mu X.\psi \mid \nu X.\psi$$

where $p \in AP$ and $X \in Var$. We will let $\mathrm{sub}(\varphi)$ denote the set of subformulas of φ. The universal fragment of μ-calculus, denoted $\Box L_\mu$, contains only formulas that do not contain the existential branching operator $\Diamond$; similarly, the existential fragment $\Diamond L_\mu$ contains only formulas that do not use the universal branching operator $\Box$.

The *least and greatest fixpoint operators*, μX and νX respectively, *bind* occurrences of the variable X in formulas of the form $\mu X.\varphi$ and $\nu X.\varphi$ (respectively). An occurrence of a variable X which is not bound by any fixpoint operator is said to be *free*, and the variable X itself is said to be free in a formula φ if it has at least one free occurrence in φ. We will say that a formula is *closed* if it contains no free variables. In addition, a closed formula is said to be *well-named* if each variable in the formula is only bound once. Since variable renaming does not change the semantics of a formula, we will only concern ourselves with well-named formulas.

Let $\mathrm{fp}_\varphi : Var \to \mathrm{sub}(\varphi)$ be the function that maps each fixpoint variable X of φ to the body of the fixpoint formula $\sigma X.\psi$ that binds X, where $\sigma = \mu, \nu$. (fp_φ is well-defined under our assumtion that φ is well-named.) We will say that X is a *greatest fixpoint variable in* φ if $\mathrm{fp}_\varphi(X) = \nu X.\psi$, and we will say that X is a *least fixpoint variable in* φ if $\mathrm{fp}_\varphi(X) = \mu X.\psi$.

A measure of the complexity of an L_μ formula is its *alternation depth*, which corresponds to the number of alternations of least and greatest fixpoint operators in the nesting hierarchy of the formula. We use the definition from [43]. Let $<$ denote the relation "is a proper subformula of", that is, $\varphi < \psi$ iff $\varphi \in \mathrm{sub}(\psi)$ and $\varphi \neq \psi$. Given an L_μ formula φ, an *alternating μ-chain* of length ℓ in φ is a sequence

$$\varphi \geq \mu X_0.\psi_0 \geq \nu X_1.\psi_1 \geq \mu X_2.\psi_2 \geq \ldots \geq \sigma X_{\ell-1}\psi_{\ell-1}$$

where $\sigma = \mu$ or ν, and for all $i < \ell - 1$, the variable X_i occurs free in every subformula ψ such that $\psi_i \geq \psi \geq \psi_{i-1}$. A ν-chain is defined symmetrically. The alternation depth $d(\varphi)$ of a formula φ is the maximal length of an alternating μ- or ν-chain in φ.

2.2.2 The semantics of μ-calculus

L_μ formulas are interpreted over Kripke structures: the *denotation* of a formula φ in a Kripke structure is the set of states in the structure that satisfy the formula. To interpret free variables in a formula, we use an *environment*, a mapping $e : Var \to 2^S$ which assigns

a set of model states to each variable. For an environment e, let $e[X \leftarrow Q]$ denote the environment obtained from e by replacing the value e assigns to X by Q.

Given a structure M and an environment e, the denotational semantics of an L_μ formula φ is defined recursively:

$$
\begin{aligned}
&[\![p]\!]^e_M = \{s \mid p \in L(s)\} \\
&[\![\neg p]\!]^e_M = \{s \mid p \notin L(s)\} \\
&[\![X]\!]^e_M = e(X) \\
&[\![\Diamond\psi]\!]^e_M = \{s \mid \exists t : (s,t) \in R \wedge t \in [\![\psi]\!]^e_M\} \\
&[\![\Box\psi]\!]^e_M = \{s \mid \forall t : (s,t) \in R \rightarrow t \in [\![\psi]\!]^e_M\} \\
&[\![\psi_1 \vee \psi_2]\!]^e_M = [\![\psi_1]\!]^e_M \cup [\![\psi_2]\!]^e_M \\
&[\![\psi_1 \wedge \psi_2]\!]^e_M = [\![\psi_1]\!]^e_M \cap [\![\psi_2]\!]^e_M \\
&[\![\mu X.\psi]\!]^e_M \text{ is the least fixpoint of the function } \tau_\psi : 2^S \rightarrow 2^S \text{ defined by} \\
&\quad \tau_\psi(Z) = [\![\psi]\!]^{e[X \leftarrow Z]}_M \\
&[\![\nu X.\psi]\!]^e_M \text{ is the greatest fixpoint of the function } \tau_\psi : 2^S \rightarrow 2^S \text{ defined above}
\end{aligned}
$$

We will omit the structure M and the environment e in $[\![\cdot]\!]^e_M$ when they are clear from the context.

For a closed L_μ formula φ, we will say that a state $s \in S$ *satisfies* φ and denote $M, s \models \varphi$ (or simply $s \models \varphi$) if $s \in [\![\varphi]\!]^\perp_M$, where $\perp$ is the environment that maps every variable to the empty set. We will say that a Kripke structure M satisfies φ and denote $M \models \varphi$ if $M, s_0 \models \varphi$.

An alternative and equivalent characterization of fixpoints is given by

$$
\begin{aligned}
[\![\mu X.\psi]\!]^e_M &= \bigcup_i \tau^i_\psi(\emptyset) \\
[\![\nu X.\psi]\!]^e_M &= \bigcap_i \tau^i_\psi(S)
\end{aligned}
$$

where τ_ψ is defined as before, and $\tau^i_\psi(Q)$ is defined by $\tau^0_\psi(Q) = Q$ and $\tau^{i+1}_\psi(Q) = \tau_\psi(\tau^i_\psi(Q))$. For a least fixpoint $\mu X.\psi$, we will call $\tau^i_\psi(\emptyset)$ the *i-th approximant* of the least fixpoint; for a greatest fixpoint $\nu X.\psi$ the i-th approximant is $\tau^i_\psi(S)$.

Since all the logical connectives except negation are monotonic, and negation can only be applied to atomic propositions, the function τ_ψ defined above is monotonic; that is, for all $A, B \subseteq S$, if $A \subseteq B$ then $\tau_\psi(A) \subseteq \tau_\psi(B)$. The fixpoints are therefore guaranteed to exist by the Knaster-Tarski Theorem. In addition, if we have a set A such that $A \subseteq \bigcup_i \tau^i_\psi(\emptyset)$ (that is, A is an under-approximation for the least fixpoint), then $\bigcup_i \tau^i_\psi(A) = \bigcup_i \tau^i_\psi(\emptyset)$; in other words, we can start the computation from any under-approximation of the least fixpoint,

instead of starting with the empty set, and still arrive at the correct result. Similarly, if we have $\bigcap_i \tau_\psi^i(S) \subseteq A$, then $\bigcap_i \tau_\psi^i(S) = \bigcap_i \tau_\psi^i(A)$: the computation of greatest fixpoints can be initialized with any over-approximation of the greatest fixpoint. These properties give rise to the Emerson-Lei model-checking algorithm presented in the next section.

2.2.3 μ-calculus model-checking

The model-checking problem for μ-calculus asks, given a Kripke structure M and an L_μ formula φ, whether or not $M \models \varphi$. A *symbolic model-checking algorithm* decides the model-checking problem by manipulating sets of model states (instead of individual states, as an *explicit* model-checking algorithm would do) to compute $[\![\varphi]\!]_M$, and then checks if $s_0 \in [\![\varphi]\!]_M$ to determine whether $M \models \varphi$.

The symbolic model-checking algorithm of Emerson and Lei [16] is given in Alg.1. The algorithm maintains an array $A[1..m]$ of approximants to the fixpoints, where m is the number of fixpoint subformulas in φ and $A[i]$ is the current approximant to the i-th fixpoint. $A[i]$ is initialized to the empty set for least fixpoint subformulas and to the set S of all states for greatest fixpoint subformulas. The algorithm computes the denotation of each subformula recursively, according the main operator: for example, the denotation of $\psi_1 \wedge \psi_2$ is computed by taking the intersection of the denotation of ψ_1 and the denotation of ψ_2, requiring two recursive calls. When the main operator is a fixpoint operator, the algorithm computes a new approximant for the fixpoint by applying the function τ_ψ defined in the previous section iteratively, until no changes occur; however, instead of computing the values of the nested fixpoints from scratch, the algorithm re-uses previous approximants for nested fixpoints of the same type as the main operator. In particular, for alternation-free formulas — formulas in which fixpoints of different types are not inter-nested — the algorithm of [16] is linear, and each fixpoint is computed using at most n approximants, where n is the number of states in the model. For detailed discussion, see [9].

2.3 Alternating Parity Tree Automata

A *normal-form alternating parity tree automaton* [43] (referred to henceforth as 'an automaton') is a tuple $A = (AP, Q, q_0, \delta, \Omega)$, where:

- AP is a set of atomic propositions;
- Q is the set of automaton states;
- q_0 is the initial state;
- δ is an alternating transition relation, assigning to each state $q \in Q$ a transition of the form $q_1 \vee q_2$, $q_1 \wedge q_2$, $\Diamond q_1$, $\Box q_1$, p or $\neg p$, where $q_1, q_2 \in Q$ and $p \in AP$;

Algorithm 1 The Emerson-Lei Symbolic Model-Checking Algorithm

function eval(φ, e):
if $\varphi = p$ **then**
  return $\{s \mid p \in L(s)\}$
else if $\varphi = X$ **then**
  return $e[X]$
else if $\varphi = \psi_1 \wedge \psi_2$ **then**
  return eval(ψ_1, e) $\cap$ eval(ψ_2, e)
else if $\varphi = \psi_1 \vee \psi_2$ **then**
  return eval(ψ_1, e) $\cup$ eval(ψ_2, e)
else if $\varphi = \Diamond\psi$ **then**
  return $\{s \mid \exists t \in S : (s,t) \in R \wedge t \in \text{eval}(\psi, e)\}$
else if $\varphi = \Box\psi$ **then**
  return $\{s \mid \forall t \in S : (s,t) \in R \Rightarrow t \in \text{eval}(\psi, e)\}$
else if $\varphi = \mu X_i.\psi$ **then**
  for all top-level greatest fixpoint subformulas $\nu X_j.\psi'$ in ψ **do**
    $A[j] \leftarrow S$
  end for
  repeat
    $Q_{\text{old}} \leftarrow A[i]$
    $A[i] \leftarrow \text{eval}(\psi, e[X_i \leftarrow A[i]])$
  until $A[i] = Q_{\text{old}}$
  return $A[i]$
else if $\varphi = \nu X_i.\psi$ **then**
  for all top-level least fixpoint subformulas $\mu X_j.\psi'$ in ψ **do**
    $A[j] \leftarrow \emptyset$
  end for
  repeat
    $Q_{\text{old}} \leftarrow A[i]$
    $A[i] \leftarrow \text{eval}(\psi, e[X_i \leftarrow A[i]])$
  until $A[i] = Q_{\text{old}}$
  return $A[i]$
end if

- $\Omega : Q \to \mathbb{N}$ is a partial priority function which represents a *parity acceptance condition*. For an automaton state $q \in Q$, $\Omega(q)$ will be called the *priority* of q.

The automata we deal with are assumed to contain no cycles of priorityless states. We will say that an infinite sequence $\pi = q_0 q_1 \ldots \in Q^\omega$ *satisfies* Ω if the lowest priority $\Omega(q)$ of a state q that has a priority and appears infinitely often in π is even.

Universal μ-calculus properties [24] can be expressed by automata that do not have $\Diamond$-transitions. We will refer to such automata as $\Box$-automata. Similarly, existential μ-calculus properties can be expressed by $\Diamond$-automata, which have no $\Box$-transitions.

2.3.1 Acceptance of automata

Tree automata run over labeled trees. A *labeled tree* is a pair $T = (N, L)$ where $N \subseteq \mathbb{N}^*$ is a prefix-closed set of tree nodes and $L : N \to 2^{AP}$ is a labeling function. The node ε (the empty word) is the tree root, and there is an edge from node n_1 to node n_2 iff $n_2 = n_1 \cdot i$ for some $i \in \mathbb{N}$. We will use $Succ(n)$ to denote the targets of edges outgoing from n; that is, $Succ(n) = \{n \cdot i \mid i \in \mathbb{N}\} \cap N$.

The acceptance of a tree by an automaton is defined in terms of a two-player infinite game. The game positions are $N \times Q$, where N is the set of tree nodes and Q is the set of automaton states, and the initial position is (ε, q_0). The player who owns the position (n, q) and the moves available to that player are determined according to δ: player I owns positions (n, q) such that $\delta(q) = q_1 \vee q_2$ or $\delta(q) = \Diamond q_1$; player II owns positions (n, q) such that $\delta(q) = q_1 \wedge q_2$ or $\delta(q) = \Box q_1$. If $\delta(q) = q_1 \vee q_2$ or $\delta(q) = q_1 \wedge q_2$, the available moves are (n, q_1) and (n, q_2); if $\delta(q) = \Diamond q_1$ or $\delta(q) = \Box q_1$, the available moves are (m, q_1) for all $m \in Succ(n)$. A position (n, q) is winning for player I if $\delta(q) = p$ and $p \in L(n)$ or $\delta(q) = \neg p$ and $p \notin L(n)$, and winning for player II if $\delta(q) = p$ and $p \notin L(n)$ or $\delta(q) = \neg p$ and $p \in L(n)$. A play is winning for player I if it is finite and ends in a position that is winning for player I, or if it is infinite and satisfies Ω; otherwise, the play is winning for player II. Strategies are defined as usual: a *strategy* for player x is a partial function mapping finite sequences of configurations to a choice of the next configuration at every position owned by player x. A play is said to be *according to* a strategy for player x if every choice made by player x in the play conforms to the strategy. A strategy is *winning* for player x if player x wins any play she plays according to the strategy.

We will say that an automaton A *accepts* a tree T iff player I has a winning strategy for the game thus described. We say that a Kripke structure M *satisfies* A, and denote $M \models A$, if the computation tree of M is accepted by A. We will also informally say that a model state $s \in S$ satisfies an automaton state $q \in Q$ if the model M', which is identical to M except that its initial state is s, is accepted by the automaton A', which is again identical to A except that its initial state is q.

Theorem 2.3.1 ([22], [43]). *For every μ-calculus formula φ there exists an alternating parity tree automaton A_φ such that for every Kripke structure M, M satisfies A_φ iff $M \models \varphi$.*

Proof sketch ([43]). The automaton $A_\varphi = (2^{AP}, \{\langle\psi\rangle \mid \psi \in \mathrm{sub}(\varphi)\}, \langle\varphi\rangle, \delta, \Omega)$ is constructed from the parse graph of the formula. Its states are the subformulas of φ, and the initial state is $\langle\varphi\rangle$ itself.

The transitions are defined according to the main connective of the formula:

- $\delta(\langle\psi_1 \circ \psi_2\rangle) = \langle\psi_1\rangle \circ \langle\psi_2\rangle$ for $\circ \in \{\wedge, \vee\}$,
- $\delta(\langle\sigma X.\psi\rangle) = \langle\psi\rangle$ for $\sigma \in \{\mu, \nu\}$,
- $\delta(\langle X\rangle) = \langle \mathrm{fp}_\varphi(X)\rangle$,
- $\delta(\langle\circ\psi\rangle) = \circ\langle\psi\rangle$ for $\circ \in \{\Diamond, \Box\}$,
- $\delta(\langle p\rangle) = p$ and $\delta(\langle\neg p\rangle) = \neg p$.

The priority function Ω is defined such that least fixpoint formulas have an odd priority, greatest fixpoint formulas have an even priority, and states that represent fixpoints at greater alternation depths have a lower priority than states that represent fixpoints at lower alternation depths. Thus the priority function requires that if we pass through a least fixpoint an infinite number of times, that fixpoint must be nested inside a greatest fixpoint. The exact details of the construction are omitted; for the full construction, see [43].

The standard construction produces an automaton that is not in normal form, but it is easy to convert it to a normal-form automaton by eliminating unit transitions of the form $\delta(q) = q_1$. (Such transitions are generated for fixpoint formulas and variables.) A unit transition $\delta(q) = q_1$ is eliminated by removing q from the set Q of automaton states, and replacing it with q_1 whenever it appears in transitions from other states. It is easy to see that the resulting automaton is equivalent to the original one. □

Example 2.3.1. Consider the $\Diamond L_\mu$ property $\varphi = \nu X.((\mu Y.p \vee \Diamond Y) \wedge \Diamond X)$, which is equivalent to the ECTL property $EGEFp$. This property is equivalent to the automaton A_φ shown in Fig. 2.1, which was obtained from φ by the standard translation (up to renaming of the states). The transitions for the automaton, the priority function, and the subformula represented by each automaton state are presented in Table 2.1. In the drawing of the automaton, a Boolean connective next to one or more edges outgoing from the same state indicates the type of the transition from the state. We use an edge to a solid black circle to indicate a transition corresponding to an atomic proposition p or $\neg p$ for some $p \in AP$.

Because q_2 corresponds to a least fixpoint subformula, its priority is odd; q_0 corresponds to a greatest fixpoint subformula, and has an even priority. The other states correspond to non-fixpoint subformulas and their priority is undefined. Note that the fixpoint subformula

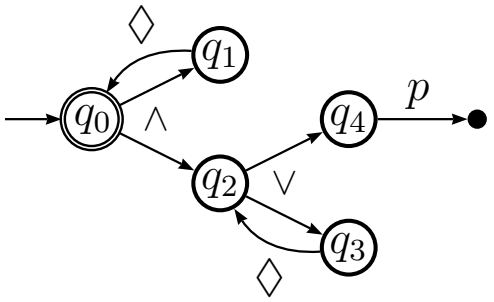

Figure 2.1. The automaton A_φ from Example 2.3.1

State	Transition	Priority	Subformula(s) represented by the state
q_0	$q_1 \wedge q_2$	0	φ, $((\mu Y.p \vee \Diamond Y) \wedge \Diamond X)$ and X
q_1	$\Diamond q_0$	undefined	$\Diamond X$
q_2	$q_3 \vee q_4$	1	$(\mu Y.p \vee \Diamond Y)$, $(p \vee \Diamond Y)$ and Y
q_3	$\Diamond q_2$	undefined	$\Diamond Y$
q_4	p	undefined	p

Table 2.1. The transitions and priorities of A_φ and the correspondence to $\mathrm{sub}(\varphi)$

$(\mu Y.p \vee \Diamond Y)$ is not nested inside $\nu X. \ldots$ in φ, because X does not occur free in $(\mu Y.p \vee \Diamond Y)$.

Note that our presentation here differs somewhat from the classical presentation of alternating automata (e.g., in [43]). In particular, the transition relation is usually represented as a function $\delta : Q \times 2^{AP} \to \mathrm{TR}$, where TR is the set of possible transitions, whereas we have $\delta : Q \to \mathrm{TR}$. Also, in [43], automata are not assumed to contain no cycles of priorityless states, and a sequence of states is said to satisfy the priority function only if it contains an infinite number of states with a defined priority.

The reason for our divergence from the accepted formalism is our particular interest in automata obtained by using the standard translation from μ-calculus formulas outlined above. Such automata have several properties which simplify their treatment in the sequel:

1. Every cycle contains a state with a defined priority: a cycle in the automaton, a "back-edge", is only created when translating a fixpoint subformula. Such cycles always contain a state representing the fixpoint itself (in the construction, each state of the automaton is a subformula), and this state will have a priority defined (even if it is a greatest fixpoint, odd if it is a least fixpoint).

2. The transition function is independent of the input to the automaton, except for states that represent atomic subformulas $p \in AP$. Formally, for all $q \in Q$, the transition function for q satisfies one of the following conditions:

   (a) For all $P_1, P_2 \in 2^{AP}$, $\delta(q, P_1) = \delta(q, P_2)$; or

(b) There is some $p \in AP$, such that for all $P \in 2^{AP}$, $\delta(q, P) = \textbf{true}$ if $p \in P$ and $\delta(q, p) = \textbf{false}$ otherwise (corresponding to the atomic formula p); or

(c) There is some $p \in AP$, such that for all $P \in 2^{AP}$, $\delta(q, P) = \textbf{false}$ if $p \in P$ and $\delta(q, p) = \textbf{true}$ otherwise (corresponding to the atomic formula $\neg p$).

We can represent the transition function for all states that satisfy the first condition as a function $\delta' : Q \to \mathrm{TR}$ (the input does not play a role). For states that satisfy the second or third condition, we add special transitions p and $\neg p$ for every $p \in AP$. The transition from a state q that satisfies the second condition will be $\delta'(q) = p$, and the transition from a state that satisfies the third condition will be $\delta'(q) = \neg p$.

In the standard translation, **true** and **false** transitions are only used in the second and third cases above; the first case corresponds to subformulas where the main connective is $\wedge$, $\vee$, $\Diamond$ or $\Box$, and as can be seen in the sketch of the proof of Theorem 2.3.1, the main connective is also the transition type. Thus, the new p and $\neg p$ transitions replace **true** and **false** and render them unnecessary. The transition types in our formalism are $\{\wedge, \vee, \Diamond, \Box, p, \neg p\}$. Acceptance for the new p and $\neg p$ transitions is defined in a way that preserves the semantics of the automaton.

These assumptions are not necessary for the encodings we present: it is easy to generalize the encodings to automata that do not enjoy the aforementioned properties. However, these assumptions are reasonable when considering practical model-checking applications, and they allow us to simplify and improve the encoding.

2.3.2 Weak automata

In Section 3.3, we present an encoding that is specialized for weak automata, which are alternating automata whose transition graph exhibits a special structure. Weak automata are equivalent in expressive power to alternation-free μ-calculus properties [26].

Formally, an automaton $A = (AP, Q, q_0, \delta, F)$ is a *weak alternating Büchi tree automaton* if there exists a partition $P = \{Q_1, \ldots, Q_n\}$ of Q, and a partial order $<_Q$ on P, satisfying the following conditions.

1. For all $1 \leq i \leq n$, either $Q_i \subseteq F$, in which case we call Q_i an *accepting set*, or $Q_i \cap F = \emptyset$, in which case Q_i is called a *rejecting set*.

2. For all $1 \leq i \leq n$ and $q \in Q_i$, if a state q' occurs in the transition from q, then $q' \in Q_j$ such that $Q_j \leq_Q Q_i$.

As for alternating parity tree automata, the acceptance of a tree by a weak alternating Büchi automaton is defined by a two-player game. The game positions and moves are the same as for alternating parity tree automata (Section 2.3.1), but the winning criterion for

infinite plays is different. In a parity automaton, Player I wins an infinite play if the sequence of states that appear in the play satisfies the parity condition. For Büchi automata, Player I wins an infinite play if it satisfies a *Büchi acceptance condition*: at least one state from F must appear infinitely often in the play.

Since every move in the play either stays in the same set of P or moves to a set that is smaller in the partial order $<_Q$, infinite plays eventually become trapped in one set Q_i and never leave it. If Q_i is an accepting set ($Q_i \subseteq F$), the infinite play is winning for Player I. If Q_i is a rejecting set ($Q_i \cap F = \emptyset$), the play will be winning for Player II.

2.4 Namjoshi-style Temporal Proofs

In [32], a proof system is presented for automata where the priority function is full. We will present the system from [32] and then explain how it can be extended to the case where the priority function is a partial function.

Let $M = (S, s_0, R, L)$ be a Kripke structure, and let $A = (AP, Q, q_0, \delta, \Omega)$ be a normal-form alternating parity tree automaton with Ω defined for all $q \in Q$. To show that M satisfies A, one must exhibit: (i) for each automaton state $q \in Q$, a predicate I_q, which, intuitively, characterises the set of model states which satisfy q; (ii) non-empty, well-founded sets $W_1, \ldots, W_m$, where m is the number of odd priorities assigned by Ω to states from Q, and pre-orders $\prec_1, \ldots, \prec_m$; (iii) for each automaton state $q \in Q$, a partial rank function $\rho_q : S \to (W, \prec)$, where $W = W_1 \times \ldots \times W_m$ and $\prec$ is the lexicographic order induced by $\prec_1, \ldots, \prec_m$ on W. In this paper, we will assume without loss of generality that $W = \mathbb{N}^m$, with $\prec_i$ the standard order $<$ over $\mathbb{N}$. We will henceforth omit W and simply write $\Pi = (I, \rho)$, where $I = \{I_q \mid q \in Q\}$ is the set of invariants and $\rho = \{\rho_q \mid q \in Q\}$ is the set of rank functions.

We use Invariance and Progress obligations to ensure that player I has a winning strategy for the game induced by A on the computation tree of M: the obligation for automaton states q with $\vee$- or $\Diamond$-transitions represents the move player I must make in positions (s, q) owned by her. Obligations for states q with $\wedge$- or $\Box$-transitions ensure that no matter which move player II makes from a position (n, q), player I will have a winning strategy from the resulting position. In the case of an infinite play, we use ranks to ensure that the play satisfies Ω.

Intuitively, the rank $\rho_q(s)$ represents a commitment regarding the number of times we may pass through states with each odd priority before passing through a state with lower priority in a play from position (n, q), where n is a tree node corresponding to model state s. For example, coordinate 0 of the rank counts the number of times we may pass through states with priority 1 before passing through a state with priority 0. Each time we pass through a state with priority $2i + 1$, coordinates 0 through i decrease lexicographically, and can only increase again when passing through a state q with $\Omega(q) < 2i+1$. A play according

to the strategy induced by the invariants can only pass through a state with an odd priority $2i+1$ a finite number of times before coordinates $0,\ldots,i$ of the rank reach zero, and then we must pass through a state with lower priority. The lowest priority occurring infinitely often in the play must be even, and player I wins.

This notion is captured by an order $\lhd_q$ over $\mathbb{N}^m$, defined for each $q \in Q$ as follows: $(x_0,\ldots,x_{m-1}) \lhd_q (y_0,\ldots,y_{m-1})$ iff $\Omega(q)=0$, or $\Omega(q)=2i, i>0$ and $(x_0,\ldots,x_i,0,\ldots,0) \preceq (y_0,\ldots,y_i,0,\ldots,0)$, or $\Omega(q)=2i+1$ and $(x_0,\ldots,x_i,0,\ldots,0) \prec (y_0,\ldots,y_i,0,\ldots,0)$. Note that coordinates $i,\ldots,m-1$ are unconstrained when passing through a state with priority $\Omega(q) < 2i+1$, but when passing through states with $\Omega(q) \geq 2i+1$, coordinate i may not increase. When passing through a state with priority $2i+1$, coordinates $0,\ldots,i$ must decrease in lexicographic order.

A valid proof must satisfy the following requirements.

- Consistency: for each $q \in Q$ and $s \in I_q$, $\rho_q(s)$ is defined.
- Initiality: $s_0 \in I_{q_0}$.
- Invariance and Progress: for each $q \in Q$ and $s \in I_q$:
  - If $\delta(q) = p$ then $p \in L(s)$.
  - If $\delta(q) = \neg p$ then $p \notin L(s)$.
  - If $\delta(q) = q_1 \vee q_2$, then either $s \in I_{q_1}$ and $\rho_{q_1}(s) \lhd_q \rho_q(s)$, or $s \in I_{q_2}$ and $\rho_{q_2}(s) \lhd_q \rho_q(s)$.
  - If $\delta(q) = q_1 \wedge q_2$, then $s \in I_{q_1}$ and $\rho_{q_1}(s) \lhd_q \rho_q(s)$, and also $s \in I_{q_2}$ and $\rho_{q_2}(s) \lhd_q \rho_q(s)$.
  - If $\delta(q) = \Diamond q_1$, then there exists $t \in S$ such that $(s,t) \in R$ and $t \in I_{q_1}$ and $\rho_{q_1}(t) \lhd_q \rho_q(s)$.
  - If $\delta(q) = \Box q_1$, then for all $t \in S$ such that $(s,t) \in R$, $t \in I_{q_1}$ and $\rho_{q_1}(t) \lhd_q \rho_q(s)$.

Theorem 2.4.1 ([32]). *For every Kripke structure M and automaton A with a full priority function, M satisfies A iff there exists a Namjoshi-style proof showing that M satisfies A.*

Automata resulting from the standard translation for μ-calculus have a *partial* priority function, with infinitely many priorities on every infinite path. For such automata, we would still like the $\lhd_q$ relation to enforce the parity acceptance condition, which now concerns only states that have a priority. Define an extension $\circledcirc_q$ as follows: $x \circledcirc_q y$ iff $\Omega(q)$ is defined and $x \lhd_q y$, or $\Omega(q)$ is undefined and $x \preceq y$.

Lemma 2.4.1. *The proof system obtained by replacing $\lhd_q$ with $\circledcirc_q$ is sound and complete for all automata with no cycles of priorityless states.*

Proof sketch. Let A be an automaton with a partial priority function Ω. We can construct an automaton with a full priority function Ω' as follows: let m be the maximal priority

assigned by Ω to any automaton state; for all priorityless states q (with $\Omega(q)$ undefined), we set $\Omega'(q) = 2m + 2$. (We need to use an even priority that is greater or equal to all the priorities assigned by Ω.) For a state q that has a priority, that is $\Omega(q)$ is defined, we set $\Omega'(q) = \Omega(q)$.

Any proof that is valid w.r.t. the $\circledcirc_q$ relation and Ω is also a valid proof w.r.t. the $\lhd_q$ relation and Ω', because if q is a state for which $\Omega(q)$ is undefined, then since $\Omega'(q)$ is an even priority greater or equal to all the other priorities, $\lhd_q$ requires a non-increase in all the coordinates of the rank. The inverse is also true: any proof that is valid w.r.t. $\lhd_q$ and Ω' is also valid w.r.t. $\circledcirc_q$ and Ω.

Let A' denote the automaton obtained from A by replacing Ω with Ω'. We argue that A and A' are equivalent.

Every play in A is also a legal play in A', and vice-versa. Therefore a legal strategy for Player I in A is also a legal strategy for Player I in A' and vice-versa. For finite plays, the winning criterion involves only the transition $\delta(q)$ and the label $L(n)$ in the last position (n, q). Since A and A' have the same transitions, every finite play is winning for Player I in A iff it is winning for Player I in A'.

Now consider an infinite play $\pi = (n_0, q_0)(n_1, q_1)\ldots$. Let $\Omega(\pi)$ denote the sequence $\Omega_0\Omega_1\ldots$ of priorities that occur in π w.r.t. Ω, defined by $\Omega_i = \Omega(q_i)$ if $\Omega(q_i)$ is defined and $\Omega_i = \perp$ if Ω is not defined for q_i. Similarly let $\Omega'(\pi) = \Omega'_0\Omega'_1\ldots$.

From the definition of Ω', one of the following conditions holds for all $i \in \mathbb{N}$:

1. $\Omega_i \neq \perp$ and $\Omega'_i = \Omega_i$; or

2. $\Omega_i = \perp$ and $\Omega'_i = 2m + 2$.

Let $\inf(\Omega(\pi))$ denote the set of priorities that appear infinitely often in $\Omega(\pi)$, excluding $\perp$, and define $\inf(\Omega'(\pi))$ similarly. From the observation above, $\inf(\Omega(\pi)) \subseteq \inf(\Omega'(\pi)) \subseteq \inf(\Omega(\pi)) \cup \{2m + 2\}$. Also, since we assumed that there are no cycles of priorityless states, there are infinitely many priorities that are not $\perp$ in $\Omega(\pi)$, and therefore $\inf(\Omega(\pi)) \neq \emptyset$ and $\min(\inf(\Omega(\pi)))$ is defined.

Because $\inf(\Omega(\pi)) \subseteq \inf(\Omega'(\pi)) \subseteq \inf(\Omega(\pi)) \cup \{2m + 2\}$ we have that $\min(\inf(\Omega(\pi))) \geq \min(\inf(\Omega'(\pi))) \geq \min(\inf(\Omega(\pi)) \cup \{2m + 2\})$. But m is the maximal priority assigned by Ω, and therefore $2m + 2 > \min(\inf(\Omega(\pi)))$ and $\min(\inf(\Omega(\pi))) = \min(\inf(\Omega(\pi)) \cup \{2m + 2\})$. We obtain that $\min(\inf(\Omega(\pi))) = \min(\inf(\Omega'(\pi))) = \min(\inf(\Omega(\pi)) \cup \{2m + 2\})$.

In other words, the minimal priority that appears infinitely often in $\Omega(\pi)$ is also the minimal priority that appears infinitely often in $\Omega'(\pi)$, and it follows that π satisfies the parity acceptance condition w.r.t. Ω iff it satisfies the parity acceptance condition w.r.t. Ω'. Hence, an infinite play in A is winning for Player I iff it is winning for Player I in A'.

From the analysis above we conclude that there exists a winning strategy for Player I in A iff that same strategy is winning for Player I in A', and thus, A and A' are equivalent.

From the fact that every valid proof w.r.t. $\circledcirc_q$ showing that $M \models A$ is also a valid proof for $M \models A'$ w.r.t. $\lhd_q$, we obtain soundness; from the inverse we obtain completeness. $\square$

Example 2.4.1. Consider the automaton A_φ from Example 2.3.1 and the model M shown in Fig. 2.2. There exists a Namjoshi-style proof Π witnessing that M is accepted by A_φ. The

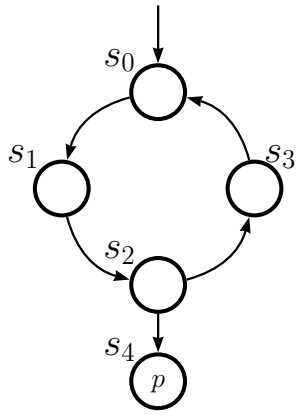

Figure 2.2. The model from Example 2.4.1

invariants and ranks of the proof are shown in Table 2.2. Note that because Ω only assigns one odd priority, each rank in Π is a vector of length one — in other words, a natural number. The $\circledcirc_q$ relation is interpreted as follows: for q_0, which has $\Omega(q_0) = 0$, $\circledcirc_{q_0}$ requires a non-increase in all the coordinates smaller than 0, of which there are none; therefore, $\circledcirc_{q_0} = \mathbb{N}^2$. For q_2, which has $\Omega(q_2) = 1$, $\circledcirc_{q_2}$ requires a decrease in the first coordinate; because there is only one coordinate, $\circledcirc_{q_2} = (<)$. For all other automaton states $q \neq q_0, q_2$, $\circledcirc_q$ requires a non-increase in all coordinates; therefore, $\circledcirc_q = (\leq)$. The relation $\circledcirc_q$ is also indicated for each automaton state in Table 2.2.

State (q)	Invariant (I_q)	Rank function (ρ_q)	$\circledcirc_q$
q_0	$\{s_0, s_1, s_2, s_3\}$	$\rho_{q_0}(s_0) = 4, \rho_{q_0}(s_1) = 3, \rho_{q_0}(s_2) = 2, \rho_{q_0}(s_3) = 5$	$\mathbb{N}^2$
q_1	$\{s_0, s_1, s_2, s_3\}$	$\rho_{q_1}(s_0) = 4, \rho_{q_1}(s_1) = 3, \rho_{q_1}(s_2) = 2, \rho_{q_1}(s_3) = 5$	$\leq$
q_2	$\{s_0, s_1, s_2, s_3, s_4\}$	$\rho_{q_2}(s_0) = 4, \rho_{q_2}(s_1) = 3, \rho_{q_2}(s_2) = 2, \rho_{q_2}(s_3) = 5, \rho_{q_2}(s_4) = 1$	$<$
q_3	$\{s_0, s_1, s_2, s_3\}$	$\rho_{q_3}(s_0) = 3, \rho_{q_3}(s_1) = 2, \rho_{q_3}(s_2) = 1, \rho_{q_3}(s_3) = 4$	$\leq$
q_4	$\{s_4\}$	$\rho_{q_4}(s_4) = 0$	$\leq$

Table 2.2. The proof Π from Example 2.4.1

Let us verify that Π is valid. First, Initiality is satisfied: $s_0 \in I_{q_0}$. Consistency is also satisfied, because for all $q \in Q$ and $s \in I_q$, $\rho_q(s)$ is defined. For Invariance and Progress, we must consider each automaton state q and each model state $s \in I_q$.

- For q_0: since $\delta(q_0) = q_1 \wedge q_2$ and $\circledcirc_{q_0} = \mathbb{N}^2$, our requirements for all $s \in I_{q_0}$ are $s \in I_{q_1}$ and $s \in I_{q_2}$. (There is no requirement on the ranks, because $\circledcirc_{q_0} = \mathbb{N}^2$.) This is satisfied, because $I_{q_1} = I_{q_0}$ and $I_{q_2} \supseteq I_{q_0}$.

- For q_1: since $\delta(q_1) = \Diamond q_0$, our requirement for all $s \in I_{q_1}$ is that there exist $t \in I_{q_0}$ with $\rho_{q_1}(t) \leq \rho_{q_0}(s)$ such that there is a transition from s to t in M. This is satisfied, because for all $0 \leq i \leq 3$ we have that $s_{(i+1) \mod 4} \in I_{q_0}$, $\rho_{q_0}(s_{(i+1) \mod 4}) = \rho_{q_1}(s_i)$ and there is a transition from s_i to $s_{(i+1) \mod 4}$.

- For q_2: since $\delta(q_2) = q_3 \vee q_4$, our requirement for all $s \in I_{q_2}$ is that either $s \in I_{q_3}$ and $\rho_{q_3}(s) < \rho_{q_2}(s)$, or $s \in I_{q_4}$ and $\rho_{q_4}(s) < \rho_{q_2}(s)$. This is satisfied: for all $s \neq s_4$ we have $s \in I_{q_3}$ and $\rho_{q_3}(s) = \rho_{q_2}(s) - 1 < \rho_{q_2}(s)$; and for s_4 we have $s_4 \in I_{q_4}$ and $\rho_{q_4}(s_4) = 0 < \rho_{q_2}(s_4) = 1$.

- For q_3: since $\delta(q_3) = \Diamond q_2$, our requirement for all $s \in I_{q_3}$ is that there exist $t \in I_{q_2}$ with $\rho_{q_2}(t) \leq \rho_{q_3}(s)$ such that there is a transition from s to t in M. This is satisfied: for $s \in \{s_0, s_1, s_3\}$ we have that $s_{(i+1) \mod 4} \in I_{q_2}$, $\rho_{q_2}(s_{(i+1) \mod 4}) = \rho_{q_3}(s_i)$ and there is a transition from s_i to $s_{(i+1) \mod 4}$. For s_2, there is a transition from s_2 to s_4, and $s_4 \in I_{q_2}$ and $\rho_{q_2}(s_4) = \rho_{q_3}(s_2) = 0$.

- For q_4: the only state in I_{q_4} is s_4. Because $\delta(q_4) = p$, we require that $p \in L(s_4)$, and this is satisfied.

2.5 Notation and Terminology

We will let $Q_\Diamond$ denote the set of automaton states q with a transition $\delta(q) = \Diamond q_1$. For automaton states $q \in Q_\Diamond$ and model states $s \in I_q$, it will sometimes be useful to identify the model state (or one of the model states) $t \in S$ which serves to satisfy the Invariance and Progress obligation for s and q in a proof Π. We will refer to t as a *proof successor* for s as required by q. (There may be more than one proof successor.)

We will let $\text{domain}(\Pi) = \bigcup_{q \in Q} I_q$. We say that the states in $\text{domain}(\Pi)$ are states that *participate* in Π.

Finally, an automaton state $q \in Q$ will be called a *terminal state* if $\delta(q) = p$ or $\delta(q) = \neg p$ for some $p \in AP$.

Chapter 3

Graph-Based BMC Encodings for Universal μ-Calculus

In this section we present our approach to reducing the bounded model-checking problem for $\Box$-automata to a SAT problem. First we present a naive translation based directly on Namjoshi-style proof obligations, and then we derive a specialized encoding for weak automata. We also discuss several practical improvements to the encodings.

We refer to our encodings as *graph-based* because of their structure: the counter-example being sought is encoded as a graph comprising k model states, where k is the bound. We do not impose restrictions on the structure of the counter-example graph, except those restrictions that follow from the semantics of the formula we are trying to disprove. Contrast with the encoding of [42], which imposes a tree-like structure on the counter-example, and the encoding of [33], which searches for a counter-example in the form of a collection of paths. See Chapter 5 for a more detailed comparison.

To constrain the k model states to represent a counter-example to the formula, we use *local obligations*. These are essentially the proof obligations in a Namjoshi-style proof for $N \models A$, where N is the model represented by our k states, and A is the automaton obtained from the negation of the formula.

3.1 Encoding Namjoshi-style Proof Obligations

The first encoding we present is a direct translation of the proof obligations of a Namjoshi-style temporal proof to Boolean constraints.

Let $M = (S, s_0, R, L)$ be a Kripke structure, represented by Boolean formulas I, R, and L_p for each $p \in AP$, as outlined in Section 2.1. Let $A = (AP, Q, q_0, \delta, \Omega)$ be a $\Diamond$-automaton representing the negation of the formula we want to model-check. To encode the requirements on ranks, we use a set of propositional formulas P_q for all $q \in Q$ ('P' stands for 'Progress'), such that $\mathsf{P}_q(\sigma_1, \sigma_2)$ holds iff $\sigma_1 \circledcirc_q \sigma_2$.

The encoding uses the following variables.

- $u_0, \ldots, u_{k-1}$: vectors representing model states. Each vector comprises n bits.
- x_i^q for each $i = 0, \ldots, k-1$ and $q \in Q$: an indicator variable for the fact that the state assigned to u_i satisfies q.
- ρ_i^q for each $i = 0, \ldots, k-1$: a vector representing the rank assigned to u_i by q. That is, if u_i is assigned the model state s, then ρ_i^q represents $\rho_q(s)$.

Each rank vector ρ_i^q has m coordinates, where m is the number of odd priorities assigned by Ω to automaton states, and each coordinate j comprises $\log |Q|k$ bits. Intuitively, this number of bits is sufficient to represent the values each coordinate may take, because if there exists an infinite winning play for player I, then there exists a play that does not pass through any odd-priority state twice before passing through a state with a lower priority. This is argued more formally in the proof of Theorem 3.1.1.

The obligations for a state u_i and an automaton state q are encoded as a Boolean formula of the form $x_i^q \rightarrow \langle\!\langle \delta(q) \rangle\!\rangle_i$, where $\langle\!\langle \delta(q) \rangle\!\rangle_i$ is defined as follows.

$$
\begin{aligned}
\langle\!\langle p \rangle\!\rangle_i &= \mathsf{L}_p(u_i) \\
\langle\!\langle \neg p \rangle\!\rangle_i &= \neg \mathsf{L}_p(u_i) \\
\langle\!\langle q_1 \wedge q_2 \rangle\!\rangle_i &= x_i^{q_1} \wedge \mathsf{P}_q(\rho_i^{q_1}, \rho_i^q) \wedge x_i^{q_2} \wedge \mathsf{P}_q(\rho_i^{q_2}, \rho_i^q) \\
\langle\!\langle q_1 \vee q_2 \rangle\!\rangle_i &= (x_i^{q_1} \wedge \mathsf{P}_q(\rho_i^{q_1}, \rho_i^q)) \vee (x_i^{q_2} \wedge \mathsf{P}_q(\rho_i^{q_2}, \rho_i^q)) \\
\langle\!\langle \Diamond q_1 \rangle\!\rangle_i &= \bigvee_{j=0}^{k-1} \left(\mathsf{R}(u_i, u_j) \wedge x_j^{q_1} \wedge \mathsf{P}_q(\rho_j^{q_1}, \rho_i^q) \right)
\end{aligned}
$$

The total number of variables used in the encoding is $O(nk + k|Q| + |Q|mk \log |Q|k)$, where n is the number of bits needed to encode one model state, k is the bound, and $|Q|$ is the number of states in the automaton. It is possible to eliminate the indicators x_i^q when $\delta(q) = q_1 \wedge q_2$ or $\delta(q) = q_1 \vee q_2$ by substituting the constraints generated for these formulas anywhere that the indicator appears; in our prototype implementation we have done so, obtaining a SAT formula with less variables. For the sake of simplicity we will retain these indicators in this presentation.

To represent the Initiality requirement, we add the constraint $\mathsf{I}(u_0) \wedge x_0^{q_0}$. The resulting formula is given by

$$
\mathsf{PRF}_{M,A,k} = \mathsf{I}(u_0) \wedge x_0^{q_0} \wedge \bigwedge_{i=0}^{k-1} \bigwedge_{q \in Q} x_i^q \rightarrow \langle\!\langle \delta(q) \rangle\!\rangle_i
$$

The scheme for using this formula is presented in Alg. 2. The BMC procedure takes a Kripke structure M, assumed to contain $|S|$ states, and a $\Box L_\mu$ formula φ. It returns **false** if it has

found a counter-example witnessing that $M \not\models \varphi$, and **true** otherwise. The procedure calls a SAT solver, which is assumed to return **sat** if the formula is satisfiable and **unsat** otherwise. In addition, the function to-automaton(ψ) is assumed to return an alternating parity tree automaton equivalent to the L_μ formula ψ, using the translation outlined in Proof 2.3.1.

The termination criterion used in Alg. 2 is extremely naive — the algorithm only halts when it has failed to find a counter-example the size of the entire model. In Section 4 we discuss alternative termination criteria, which may halt the bounded model-checking well before this coarse bound is reached. Note that in traditional linear-time BMC encodings, the size of the model is not a sound completeness threshold; in our encoding it is a sound threshold, and this is one distinguishing feature of our encoding. This is discussed further in Section 5.4. The soundness of the algorithm is proven in Theorem 3.1.1.

Algorithm 2 Bounded Model-Checking Using the Graph-Based Encoding

```
function bmc(M, φ)
A ← to-automaton(¬φ)
for k = 1 to |S| do
  res ← SAT-solve(PRF_{M,A,k})
  if res = sat then
    return false
  end if
end for
return true
```

Theorem 3.1.1 (Correctness)**.** *For all Kripke structures M and $\square L_\mu$ formulas φ, Alg. 2 returns* **true** *iff $M \models \varphi$.*

Proof. To prove that Alg. 2 is correct, we must establish that $\mathsf{PRF}_{M,A,k}$ is satisfiable iff M admits a counter-example of size k to φ.

Suppose N is a counter-example of size k to $M \models \varphi$, and let $\{s_0, \ldots, s_{k-1}\}$ be the states of N. (N must include the initial state s_0 of M.)

From Theorem 2.4.1, there exists a Namjoshi-style proof $\Pi = (I, \rho)$ showing that N satisfies A, where A is the automaton returned by to-automaton($\neg\varphi$).

Suppose A has $|Q|$ states. Π contains $|Q|$ ranking functions — one ranking function $\rho_q : \{s_0, \ldots, s_{k-1}\} \to \mathbb{N}^m$ for each $q \in Q$. Each ranking function ρ_q may be a partial function or a full function, but in the worst case it can assign k different vectors from $\mathbb{N}^m$, one value for each model state s_i. In particular, each ranking function ρ_q assigns no more than k different values to each *coordinate* in the rank vector, and thus, Π uses a total of no more than $|Q|k$ values for each coordinate of the rank vector. We can assume w.l.o.g. that the range of each ranking function is $\{0, \ldots, |Q|k\}^m$ (instead of $\mathbb{N}^m$). Under this assumption, $\log |Q|k$ bits are sufficient to encode each coordinate of the rank vector.

Construct an assignment v as follows: for all $i \in \{0, \ldots, k-1\}$ and $q \in Q$,

- $v(u_i) = s_i$,
- $v(x_i^q) = 1$ if $s_i \in I_q$ and $v(x_i^q) = 0$ otherwise,
- $v(\rho_i^q) = \rho_q(s_i)$, encoded as a binary number.

The assignment v thus defined satisfies $\mathsf{PRF}_{M,A,k}$, because $\mathsf{PRF}_{M,A,k}$ is merely a Boolean encoding of the Initiality and Invariance and Progress proof obligations. Π satisfies these obligations because it is a valid proof; therefore v satisfies $\mathsf{PRF}_{M,A,k}$.

For the other direction, suppose there exists an assignment v satisfying $\mathsf{PRF}_{M,A,k}$. To show that M admits a counter-example of size k, we simply perform the reverse construction, obtaining a Namjoshi-style proof $\Pi = (I, \rho)$ from v as follows: for all $i \in \{0, \ldots, k-1\}$ and $q \in Q$,

- $v(u_i) \in I_q$ iff $v(x_i^q) = 1$;
- $\rho_q(v(u_i)) = v(\rho_i^q)$.

The proof Π thus defined satisfies Initiality and Invariance and Progress, because $\mathsf{PRF}_{M,A,k}$ encodes these obligations; in addition, Π satisfies Consistency, because the ranking functions ρ_q are defined for all the states that participate in the proof (regardless of whether or not they actually belong to the corresponding invariant I_q).

Let N be the substructure induced by $\{v(s_0), \ldots, v(s_{k-1})\}$ on M. (The transition relation of N comprises all those transitions of M that involve two states from N.) N is a counter-example of size k for $M \models \varphi$: it is a submodel of M, and it does not satisfy φ, because Π is a valid Namjoshi-style proof showing that N satisfies A.

Finally, for the correctness of the algorithm, observe that if a counter-example for $M \models \varphi$ exists, it contains no more than $|S|$ states, where $|S|$ is the number of states in M. If we have not found a counter-example in the last iteration, when $k = |S|$, we can conclude that a counter-example does not exist and $M \models \varphi$. The correctness of the algorithm follows: Alg 2 returns **true** iff it does not return **false**, which occurs iff for all $k \in \{1, \ldots, |S|\}$ the formula $\mathsf{PRF}_{M,A,k}$ is unsatisfiable; this, in turn, happens iff there is no counter-example of size k for all $k \in \{1, \ldots, |S|\}$, which by our previous argument is true iff there is no counter-example of any size, meaning $M \models \varphi$. □

3.2 Improving the Encoding

The encoding presented above is a naive translation of Namjoshi-style proof obligations to SAT. It can be improved in several ways.

3.2.1 Symmetry-breaking

The formula $\mathsf{PRF}_{M,A,k}$ suffers from symmetry, which has an adverse effect on the performance of SAT solvers (see, e.g., [1]). The model states $u_1, \ldots, u_{k-1}$ are interchangeable: intuitively, given a satisfying assignment, it is possible to permute the values of $u_1, \ldots, u_{k-1}$ and the accompanying ranks and indicator variables and obtain a different satisfying assignment. The SAT solver thus has to consider many equivalent permutations that represent the same counter-example before eliminating it.

We have found that performance is greatly improved when we break the symmetry by ordering the states $u_1, \ldots, u_{k-1}$, obtaining the formula

$$\mathsf{PRF}^{\mathsf{sym}}_{M,A,k} = \mathsf{PRF}_{M,A,k} \wedge \bigwedge_{i=1}^{k-2} u_i < u_{i+1}$$

where "$<$" is implemented as the lexicographic order on binary vectors.

The state u_0 is excluded from the ordering, because it serves a "special" role: it must be the initial state of the model. We therefore cannot require that it be lexicographically smaller than all the other states in the counter-example. The rest of the states, $u_1, \ldots, u_{k-1}$, are interchangeable, as shown by the following lemma.

Lemma 3.2.1. *The formula $\mathsf{PRF}_{M,A,k}$ is satisfiable if and only if $\mathsf{PRF}^{\mathsf{sym}}_{M,A,k}$ is satisfiable.*

Proof. The main idea of the proof is to support our claim that the states $u_1, \ldots, u_{k-1}$ are 'interchangeable' by showing how they can be interchanged without affecting the satisfaction of the formula.

Since $\mathsf{PRF}^{\mathsf{sym}}_{M,A,k}$ is the conjunction of $\mathsf{PRF}_{M,A,k}$ with another formula, if $\mathsf{PRF}^{\mathsf{sym}}_{M,A,k}$ is satisfiable then $\mathsf{PRF}_{M,A,k}$ is clearly satisfiable as well.

For the other direction, suppose there exists an assignment v for $\mathsf{PRF}_{M,A,k}$, in which each state vector u_i is assigned a model state $v(u_i) = s_i$. From the Initiality requirement in $\mathsf{PRF}_{M,A,k}$, $v(u_0) = s_0$ is the initial state of M. Let $\ell : \{1, \ldots, k-1\} \to \{1, \ldots, k-1\}$ be the permutation that orders the model states $s_1, \ldots, s_{k-1}$ lexicographically: $s_{\ell(1)} < s_{\ell(2)} < \ldots < s_{\ell(k-1)}$. Define $\ell(0) = 0$. Let ℓ^{-1} be the inverse of ℓ. ℓ and ℓ^{-1} are both permutations over $\{0, \ldots, k-1\}$.

Construct a new assignment v' as follows: for all $i \in \{0, \ldots, k-1\}$, set $v'(u_i) = v(u_{\ell(i)})$, and set all the auxiliary variables associated with u_i in v' to the values of the corresponding variables associated with $u_{\ell(i)}$ in v: for all $q \in Q$, $v'(x^q_i) = v(x^q_{\ell(i)})$ and $v'(\rho^q_i) = v'(\rho^q_{\ell(i)})$.

For a formula α, let us denote by $\ell(\alpha)$ the formula obtained by substituting $u_{\ell(i)}$, $x^q_{\ell(i)}$ and $\rho^q_{\ell(i)}$ for all occurrences of u_i, x^q_i and ρ^q_i in α, respectively, for all $i \in \{0, \ldots, k-1\}$ and $q \in Q$. It is easy to see from the construction of v' that for all α, $v \models \alpha$ iff $v' \models \ell(\alpha)$.

We will show that $v' \models \mathsf{PRF}^{\mathsf{sym}}_{M,A,k}$ by showing that v' satisfies each conjunct in $\mathsf{PRF}^{\mathsf{sym}}_{M,A,k}$.

- $\alpha = (I(u_0) \wedge x_0^{q_0})$ (Initiality): this conjunct also appears in $\mathsf{PRF}_{M,A,k}$, and therefore v satisfies it. Since v' assigns the same values as v to u_0 and $x_0^{q_0}$, v' satisfies this conjunct as well.

- $\alpha = (x_i^q \rightarrow \langle\!\langle \delta(q) \rangle\!\rangle_i)$ (Invariance and Progress for u_i and q):
  Let $\beta = \left(x^q_{\ell^{-1}(i)} \rightarrow \langle\!\langle \delta(q) \rangle\!\rangle_{\ell^{-1}(i)}\right)$. β appears in $\mathsf{PRF}_{M,A,k}$ as a conjunct, and therefore $v \models \beta$.

  If $\delta(q)$ is *not* of the form $\delta(q) = \Diamond q_1$, then from the definition of $\langle\!\langle \delta(q) \rangle\!\rangle_{\ell^{-1}(i)}$, $\ell(\beta) = \alpha$. From our observation about the relationship between v and v' we have that $v' \models \alpha$.

  However, if $\delta(q) = \Diamond q_1$ for some $q_1 \in Q$, then $\ell(\beta) \neq \alpha$: recall that

$$\langle\!\langle \Diamond q_1 \rangle\!\rangle_{\ell^{-1}(i)} = \bigvee_{j=0}^{k-1} \left(\mathsf{R}(u_{\ell^{-1}(i)}, u_j) \wedge x_j^{q_1} \wedge \mathsf{P}_q(\rho_j^{q_1}, \rho^q_{\ell^{-1}(i)})\right)$$

  and therefore

$$\ell(\beta) = \ell(x^q_{\ell^{-1}(i)} \rightarrow \langle\!\langle \Diamond q_1 \rangle\!\rangle_{\ell^{-1}(i)}) = x_i^q \rightarrow \bigvee_{j=0}^{k-1} \left(\mathsf{R}(u_i, u_{\ell(j)}) \wedge x_{\ell(j)}^{q_1} \wedge \mathsf{P}_q(\rho^{q_1}_{\ell(j)}, \rho_i^q)\right)$$

  In other words, the order in which the state vectors appear in the disjunction is different from their order in α. However, although α and $\ell(\beta)$ are not syntactically equal in this case, they are clearly semantically equivalent: since ℓ is a permutation of $\{0, \ldots, k-1\}$, all the indices appear in the disjunction, and since disjunction is commutative the order does not matter. As before, $\mathsf{PRF}_{M,A,k}$ contains β as a conjunct, which v must satisfy; hence, $v' \models \ell(\beta)$, and therefore $v' \models \alpha$ as well.

- $\left(\bigwedge_{i=1}^{k-2} u_i < u_{i+1}\right)$ (orderedness): v' satisfies this conjunct because for all $i \in \{0, \ldots, k-1\}$, $v'(u_i) = s_{\ell(i)}$, and $s_{\ell(1)} < \ldots < s_{\ell(k-1)}$.

Note that except for the last conjunct, which expresses the orderedness constraint intended to break the symmetry, our proof that $v' \models \mathsf{PRF}^{\mathsf{sym}}_{M,A,k}$ is independent of the choice of the permutation ℓ; hence our claim that $u_1, \ldots, u_{k-1}$ are interchangeable. □

3.2.2 Encoding successor states explicitly

The formula $\mathsf{PRF}_{M,A,k}$ contains a constraint expressing the Invariance and Progress obligation for each model state s_i and each automaton state $q \in Q$. For states $q \in Q_\Diamond$ with a transition $\delta(q) = \Diamond q_1$, each such constraint contains k copies of the model's transition relation R:

$$\langle\!\langle \Diamond q_1 \rangle\!\rangle_i = \bigvee_{j=0}^{k-1} \left(\mathsf{R}(u_i, u_j) \wedge x_j^{q_1} \wedge \mathsf{P}_q(\rho_j^{q_1}, \rho_i^q)\right)$$

Thus, the transition relation appears a total of $k^2|Q_\Diamond|$ times in $\mathsf{PRF}_{M,A,k}$ and its variants.

The transition relation is usually the most complicated aspect of the model; it is generally a very large formula. It is desirable to decrease the number of times it appears in $\mathsf{PRF}_{M,A,k}$. We can do so, at the cost of adding more variables. As a side-effect, we will also gain valuable information about the counter-example.

Recall that in the original formula $\mathsf{PRF}_{M,A,k}$, the following variables are used:

- $u_0, \ldots, u_{k-1}$: vectors representing model states.
- x_i^q for each $i = 0, \ldots, k-1$ and $q \in Q$: an indicator variable for the fact that the state assigned to u_i satisfies q.
- ρ_i^q for each $i = 0, \ldots, k-1$: a vector representing the rank assigned to u_i by q.

We will now add one model-state vector t_i^q for every $q \in Q_\Diamond$ and $i \in \{0, \ldots, k-1\}$, increasing the total number of variables to $O(nk + k|Q| + |Q|mk \log |Q|k + nk|Q_\Diamond|)$ (a linear increase which does not change the overall complexity). The state t_i^q will be interpreted to represent the proof-successor for u_i required by q if $x_i^q = 1$. Since we are searching for a counter-example of bounded size k, t_i^q must be assigned one of the counter-example states u_j. In addition, if $\delta(q) = \Diamond q_1$, then u_j must be in the appropriate invariant I_{q_1}, and its rank must behave appropriately. All these requirements will reflect in the Invariance and Progress requirement for u_i and q.

Define new constraints $\langle\!\langle \delta(q) \rangle\!\rangle_i^{\mathsf{exp}}$ as follows:

- If $q \notin Q_\Diamond$, then $\langle\!\langle \delta(q) \rangle\!\rangle_i^{\mathsf{exp}} = \langle\!\langle \delta(q) \rangle\!\rangle_i$.
- If $q \in Q_\Diamond$ and $\delta(q) = \Diamond q_1$, then

$$\langle\!\langle \delta(q) \rangle\!\rangle_i^{\mathsf{exp}} = \langle\!\langle \Diamond q_1 \rangle\!\rangle_i^{\mathsf{exp}} = \mathsf{R}(u_i, t_i^q) \wedge \bigvee_{j=0}^{k-1} \left(t_i^q = u_j \wedge x_j^{q_1} \wedge \mathsf{P}_q(\rho_j^{q_1}, \rho_i^q) \right)$$

Now define a formula $\mathsf{PRF}_{M,A,k}^{\mathsf{exp}}$ which uses the new constraints:

$$\mathsf{PRF}_{M,A,k}^{\mathsf{exp}} = \mathsf{I}(u_0) \wedge x_0^{q_0} \wedge \bigwedge_{i=0}^{k-1} \bigwedge_{q \in Q} x_i^q \rightarrow \langle\!\langle \delta(q) \rangle\!\rangle_i^{\mathsf{exp}}$$

In the new formula, the transition relation appears $k \cdot |Q_\Diamond|$ times instead of $k^2 \cdot |Q_\Diamond|$ times as before. We will also have further use for the information we gain by explicitly encoding proof successors in constructing a counter-example (Section 3.4) and in constructing a dynamic completeness criterion (Section 4.2).

Lemma 3.2.2. *$\mathsf{PRF}_{M,A,k}$ is satisfiable iff $\mathsf{PRF}_{M,A,k}^{\mathsf{exp}}$ is satisfiable.*

Proof. Suppose $\mathsf{PRF}_{M,A,k}$ is satisfiable, and let v be a satisfying assignment. In particular, for all $q \in Q_\Diamond$ and $i \in \{0,\ldots,k-1\}$, v must satisfy

$$x_i^q \to \bigvee_{j=0}^{k-1} \left(\mathsf{R}(u_i,u_j) \wedge x_j^{q_1} \wedge \mathsf{P}_q(\rho_j^{q_1},\rho_i^q)\right)$$

Define a mapping $\gamma : \{0,\ldots,k-1\} \to \{0,\ldots,k-1\}$ as follows: for $i \in \{0,\ldots,k-1\}$, if $v(x_i^q) = 0$ set $\gamma(u) = 0$. Otherwise, because v satisfies the constraint above there exists some $j \in \{0,\ldots,k-1\}$ such that $v \models \left(\mathsf{R}(u_i,u_j) \wedge x_j^{q_1} \wedge \mathsf{P}_q(\rho_j^{q_1},\rho_i^q\right)$. In this case define $\gamma(i) = j$. (If there is more than one choice for j, choose arbitrarily among the viable options.)

Construct an assignment v' for $\mathsf{PRF}^{\mathsf{exp}}_{M,A,k}$ as follows: for all $i \in \{0,\ldots,k-1\}$ and $q \in Q$,

- $v'(u_i) = v(u_i)$,
- $v'(x_i^q) = v(x_i^q)$,
- $v'(\rho_i^q) = v(\rho_i^q)$,
- $v'(t_i^q) = v(u_{\gamma(i)})$.

In $\mathsf{PRF}^{\mathsf{exp}}_{M,A,k}$, all constraints except constraints of the form $x_i^q \to \langle\!\langle \delta \rangle\!\rangle_i^{\mathsf{exp}}$ for some $q \in Q$ are identical to the constraints that appear in $\mathsf{PRF}_{M,A,k}$, and involve only variables that get the same value under v' as they do under v. These constraints are satisfied under v' as they are under v.

The constraints that remain are of the form

$$x_i^q \to \mathsf{R}(u_i,t_i^q) \wedge \bigvee_{j=0}^{k-1} \left(t_i^q = u_j \wedge x_j^{q_1} \wedge \mathsf{P}_q(\rho_j^{q_1},\rho_i^q)\right)$$

and they are also satisfied: if $v(x_i^q) = 0$ then $v'(x_i^q) = 0$ as well, and the implication is satisfied; otherwise, from the definition of $\gamma(i)$, $v \models \left(\mathsf{R}(u_i,u_{\gamma(i)}) \wedge x_{\gamma(i)}^{q_1} \wedge \mathsf{P}_q(\rho_{\gamma(i)}^{q_1},\rho_i^q\right)$. Therefore, since $v'(t_i^q) = v(u_{\gamma(i)})$, $v' \models \mathsf{R}(u_i,t_i^q)$, and there exists $j = \gamma(i)$ such that $v' \models \left(t_i^q = u_j \wedge x_j^{q_1} \wedge \mathsf{P}_q(\rho_j^{q_1},\rho_i^q)\right)$. Together we have that v' satisfies the entire constraint.

Now suppose that $\mathsf{PRF}^{\mathsf{exp}}_{M,A,k}$ is satisfiable and v is a satisfying assignment. We argue that $v \models \mathsf{PRF}_{M,A,k}$ as well. The only constraints that appear in $\mathsf{PRF}_{M,A,k}$ but not in $\mathsf{PRF}^{\mathsf{exp}}_{M,A,k}$ are of the form

$$\alpha = x_i^q \to \bigvee_{j=0}^{k-1} \left(\mathsf{R}(u_i,u_j) \wedge x_j^{q_1} \wedge \mathsf{P}_q(\rho_j^{q_1},\rho_i^q)\right)$$

However, for each such constraint, $\mathsf{PRF}^{\mathsf{exp}}_{M,A,k}$ contains the constraint

$$\beta = x_i^q \to \mathsf{R}(u_i,t_i^q) \wedge \bigvee_{j=0}^{k-1} \left(t_i^q = u_j \wedge x_j^{q_1} \wedge \mathsf{P}_q(\rho_j^{q_1},\rho_i^q)\right)$$

which v satisfies. Hence, either $v(x_i^q) = 0$, in which case $v \models \alpha$ too, or there exists some $j \in \{0, \ldots, k-1\}$ such that $v(t_i^q) = v(u_j)$, and also $v(x_j^{q_1}) = 1$ and $v \models \mathsf{P}_q(\rho_j^{q_1}, \rho_i^q)$. It follows that $v \models \mathsf{R}(u_i, u_j)$. We have shown that there exists $j \in \{0, \ldots, k-1\}$ such that $v \models \big(\mathsf{R}(u_i, u_j) \wedge x_j^{q_1} \wedge \mathsf{P}_q(\rho_j^{q_1}, \rho_i^q)\big)$; hence, v satisfies the disjunction over j in α, and $v \models \alpha$. □

3.3 A Specialized Encoding for Weak Automata

Our experiments, presented in Chapter 6, showed that the ranks and progress relation used in the encoding are difficult for SAT solvers to handle.

In our implementation we rather naively encoded the integer ranks in binary representation, and implemented the P_q relation as binary $<$ or $\leq$ (depending on the priority of q). Performance can probably be improved by finding better encodings for the P_q progress relation, and this is discussed in Chaper 7.1. In this section we suggest a variation on the encoding that eliminates the explicit use of ranks and progress relations. The encoding is tailored for weak automata; it can be extended to handle general automata, but then it becomes very inefficient.

Let $A = (AP, Q, q_0, \delta, F)$ be a weak automaton with partition $P = (Q_1, \ldots, Q_N)$ and order $<_Q$ on P, as described in Section 2.3.2. Let D be the size of the largest rejecting set in P.

Let M be a Kripke structure. If A accepts the computation tree of M, then there is a memoryless winning strategy for Player I in the game induced by A and M. When Player I plays according to this strategy, every infinite play eventually becomes trapped in an accepting set.

In a Namjoshi-style proof for $M \models A$, we do not need the complicated $\circledcirc_q$ progress relation, which is intended to ensure that the parity acceptance condition is satisfied in every infinite play. Instead, we only need to ensure that every time the play enters a rejecting set, it exits it after a finite number of moves. To do this it is sufficient to attach to each game position (s, q) with $q \in Q_i$, where Q_i is a rejecting set, a counter $\sigma_q(s)$, which limits the number of steps until the play exits Q_i. Every time we move to another state within Q_i, the counter must decrease, until it reaches zero and then we must exit Q_i. For positions (s, q) where q is in an accepting set, there is no need for a counter.

Based on this observation, we define a Namjoshi-style proof system for weak automata. Given a weak automaton A and a Kripke structure M, a weak Namjoshi-style proof that M satisfies A is a pair $\Pi = (I, \sigma)$, where $I = \{I_q \mid q \in Q\}$ is a set of invariants and $\sigma = \{\sigma_q : S \to \mathbb{N} \mid q \in Q \setminus F\}$ is a set of partial rank functions for rejecting states.

Given two states $q, q' \in Q$, we define a relation $<_{q \to q'} \subseteq \mathbb{N}^2$ as follows:

- If $q' \in F$, then for all $q \in Q$, $<_{q \to q'} = \mathbb{N}^2$. (Intuitively, we do not care what happens to

the rank when we move between two accepting states, because the play is allowed to stay in an accepting set forever.)

- If q and q' are not in the same set, then again $<_{q \to q'} = \mathbb{N}^2$. (Intuitively, we do not care what happens to the rank when we move between sets; we only need to count the number of steps inside a set.)

- If q, q' are in the same set and it is rejecting, then $<_{q \to q'} = \{(x, y) \in \mathbb{N}^2 \mid x < y\}$, where $<$ is the natural order on integers.

A valid weak proof must satisfy the following obligations.

- Consistency: for each $q \in Q \setminus F$ and $s \in I_q$, $\sigma_q(s)$ is defined.

- Initiality: $s_0 \in I_{q_0}$.

- Invariance and Progress: for each $q \in Q$ and $s \in I_q$:

  - If $\delta(q) = p$ then $p \in L(s)$.
  - If $\delta(q) = \neg p$ then $p \notin L(s)$.
  - If $\delta(q) = q_1 \vee q_2$, then either $s \in I_{q_1}$ and $\sigma_{q_1}(s) <_{q \to q_1} \sigma_q(s)$, or $s \in I_{q_2}$ and $\sigma_{q_2}(s) <_{q \to q_2} \sigma_q(s)$.
  - If $\delta(q) = q_1 \wedge q_2$, then $s \in I_{q_1}$ and $\sigma_{q_1}(s) <_{q \to q_1} \sigma_q(s)$, and also $s \in I_{q_2}$ and $\sigma_{q_2}(s) <_{q \to q_2} \sigma_q(s)$.
  - If $\delta(q) = \Diamond q_1$, then there exists $t \in S$ such that $(s, t) \in R$ and $t \in I_{q_1}$ and $\sigma_{q_1}(t) <_{q \to q_1} \sigma_q(s)$.
  - If $\delta(q) = \Box q_1$, then for all $t \in S$ such that $(s, t) \in R$, $t \in I_{q_1}$ and $\sigma_{q_1}(t) <_{q \to q_1} \sigma_q(s)$.

Lemma 3.3.1 (Soundness and completeness)**.** *If A is a weak automaton, there exists a valid weak proof for $M \models A$ iff $M \models A$.*

Proof sketch. The proof is very similar to the one used in [32] to show the soundness and completeness of the original proof system. We sketch the outline of the proof, without entering into the technical details from [32].

In the first direction (soundness), if there exists a weak proof Π for $M \models A$, the original soundness proof in [32] constructs a winning strategy for Player I based on the proof. The construction works in our case as well. The Invariance and Progress obligations ensure that Player I always has a legal move to a position (s, q) with $s \in I_q$; if Player I moves from position (s, q) to position (s', q'), where q and q' are in the same rejecting set, then $\sigma_q(s)$ and $\sigma_{q'}(s')$ are both defined (Consistency) and $\sigma_{q'}(s') < \sigma_q(s)$ (Progress). Thus, after a finite number of steps within the same rejecting set, the play must exit the set and move to a

different set. Since there are no cycles in the partial order $<_Q$ on the partition P, we cannot return to a rejecting set once we have left it, and eventually the play must become trapped in an accepting set. Hence, every infinite play is winning for Player I. Every finite play is also winning for Player I: if the play reaches a terminal state, the Invariance obligations ensure that it is winning for Player I. Thus, the strategy is a winning strategy for Player I, and therefore $M \models A$.

To show relative completeness, we again rely on the proof from [32]. The main idea in the proof is to convert the question of whether Player I has a winning strategy from a given position into a model-checking problem, and use information obtained from a model-checker's run to construct the proof.

We can use the construction from [32] to show the existence of a weak proof. To convince the reader that this is the case, we sketch the construction from [32], show that the resulting Namjoshi-style proof has the form we need, and show that it satisfies the obligations of a weak proof. The sketch of the proof from [32] is not necessary; we present it in the interest of self-containment.

The proof in [32] uses μ-calculus signatures. We do not enter into a technical discussion of μ-calculus signatures here, because the technical details are not necessary for our proof. Informally, a μ-calculus signature $\mathrm{sig}(\varphi, s)$ for a formula φ and a model state s records how many unfoldings of each least fixpoint in φ are necessary to show that $s \models \varphi$. The μ-calculus signature is a vector $(x_0, \ldots, x_m)$, where m is the number of least fixpoints in φ. For our purposes it is sufficient to note that if φ contains only one least fixpoint, the μ-calculus signature $\mathrm{sig}(\varphi, s)$ is a vector with only one coordinate — in other words, $\mathrm{sig}(\varphi, s)$ is a natural number.

Suppose M satisfies an automaton A. The construction from [32] proceeds by first noting that Player I has a winning strategy from position (s, q) iff $M \times A, (s, q) \models \mathcal{W}_I$, where $M \times A$ and $\mathcal{W}_I$ are defined as follows.

- $M \times A$ is the cross product of the model M and the automaton A, defined by $M \times A = (S \times Q, (s_0, q_0), R', L')$ over the atomic propositions $AP' = \{I, II, win_I, win_{II}\} \times \{\Omega_0, \ldots, \Omega_m\}$, where m is the maximal priority assigned by Ω to any automaton state.

  The transition relation R' expresses the moves in the game: for $s \in S$ and $q \in Q$, if $\delta(q) = p$ or $\delta(q) = \neg p$, then (s, q) has no transitions in $M \times A$; if $\delta(q) = q_1 \vee q_2$ or $\delta(q) = q_1 \wedge q_2$, then $((s, q), (s, q_1)) \in R'$ and $((s, q), (s, q_2)) \in R'$; and if $\delta(q) = \Diamond q_1$ or $\delta(q) = \Box q_1$, then $((s, q), (t, q_1)) \in R'$ for all $t \in S$ such that $(s, t) \in R$.

  The first component of the labeling $L'((s, q))$ expresses which player owns position (s, q) in the game: if (s, q) is a Player I position, then the first component of $L'((s, q))$ is I; if (s, q) is a Player II position then the first component is II. If (s, q) is a terminal position, the first component of L' indicates which player wins the play: if $\delta(q) = p$

and $p \in L(s)$, or $\delta(q) = \neg p$ and $p \notin L(s)$, then the first component of $L'((s,q)$ is win_I, and if $\delta(q) = p$ and $p \notin L(s)$, or $\delta(q) = \neg p$ and $p \in L(s)$, then the first component of $L'((s,q))$ is win_{II}.

The second component of the labeling $L'((s,q))$ indicates the priority of q, and it is Ω_i iff $\Omega(q) = i$.

- $\mathcal{W}_I = \sigma_0 Z_0 \ldots \sigma_m Z_m . win_I \vee (I \wedge \bigwedge_i \Omega_i \rightarrow \Diamond Z_i) \vee (II \wedge \bigwedge_i \Omega_i \rightarrow \Box Z_i)$, where σ_i is ν if i is even and μ if i is odd.

Intuitively, $\mathcal{W}_I$ expresses the existence of a winning strategy for Player I: a position must either be terminal and winning for Player I (win_I), or it can be a Player I position, in which case Player I chooses her next move to another winning position ($\Diamond Z_i$), or it can be a Player II position, in which case Player II chooses the move, so all possible successors must be winning for Player I ($\Box Z_i$). The fixpoints express the parity condition: the play may not pass through an odd-priority position (labeled with Ω_i where i is odd, and corresponding to the least fixpoint Z_i) unless it also passes through a lower even-priority position infinitely often (a greatest fixpoint Z_j with $j < i$).

Thus, [32] translates the question of whether or not Player I has a winning strategy into a model-checking problem. Next, a valid proof for $M \models A$ is formed by taking the invariant I_q to be $I_q = \{s \in S \mid M \times A, (s,q) \models \mathcal{W}_I\}$ for all $q \in Q$, and $\rho_q(s)$ to be the μ-calculus signature of $\mathcal{W}_I$ at (s,q). Essentially, each invariant is the set of model states s such that Player I has a winning strategy from position (s,q), and the rank $\rho_q(s)$ expresses how many unfoldings of each fixpoint in $\mathcal{W}_I$ are needed to show that $M \times A, (s,q) \models \mathcal{W}_I$.

All of the above explains how to form a proof for a parity automaton. A Büchi acceptance condition with a set F of accepting states is nothing more than the parity acceptance condition Ω_F, where $\Omega_F(q) = 1$ if $q \notin F$ and $\Omega_F(q) = 0$ if $q \in F$. To satisfy Ω_F, an infinite computation has to pass infinitely often through a state with priority 0, which means it must pass infinitely often through F.

When we use this parity condition in the proof outlined above, the formula $\mathcal{W}_I$ we obtain has only two fixpoints: it is of the form $\nu Z_0 . \mu Z_1 . \psi_I$. The μ-calculus signature of $\mathcal{W}_I$ in this case is a single-coordinate vector, in other words, a number; the μ-calculus signature of $\mathcal{W}_I$ at (s,q) simply says how many times we need to unfold the least fixpoint μZ_1 to show that $\mathcal{W}_I$ is satisfied by (s,q). Therefore the valid proof $\Pi = (I, \rho)$ constructed in the relative completeness proof from [32] already has the form we need. It is also easy to see that the proof obligations are satisfied: the proof Π satisfies the obligations for general parity automata, and our obligations are very similar. The Initiality and Consistency obligations are the same, and so is the Invariance part of Invariance and Progress. As for Progress, it is sufficient to note that the $\lhd_q$ progress relation is actually stronger than the $<_{q \rightarrow q'}$ relation, which only requires a decrease in rank when we move to an automaton state in the same

rejecting set. Rejecting automaton states have odd priority in Ω_F, so $\lhd_q$ also requires a decrease in rank in the Invariance and Progress obligations involving such states. (The $\lhd_q$ relation also requires a decrease in rank when passing through *any* rejecting state, but $<_{q \to q'}$ does not.) Since Π satisfies Invariance and Progress w.r.t. the stronger relation $\lhd_q$, it also satisfies them w.r.t. $<_{q \to q'}$. □

Using a single-coordinate counter with a simple progress relation is a clear improvement over the encoding for non-weak automata. However, it still uses ranks, albeit simpler ones. To eliminate the use of explicit ranks, we use ideas from [19].

Recall that in the original encoding we used a bit x_i^q to indicate whether or not u_i satisfies q. If q is in a rejecting set, instead of encoding the rank assigned by q to u_i explicitly as a binary vector ρ_i^q, we will have one copy of x_i^q for each possible value of $\sigma_q(s)$. An assignment of 1 to the t-th copy, $x_i^{q,t}$, will be interpreted to mean that u_i satisfies q and we leave q's rejecting set within at most t steps in the play. For states q that belong to an accepting set ($q \in F$), we will not require more than one copy. We will denote this single copy by $x_i^{q,N-1}$.

Suppose we are using N copies $x_i^{q,0}, \ldots, x_i^{q,N-1}$ of each bit x_i^q where q is in a rejecting set. (The value of N will be specified in the next paragraph.) Define a Boolean formula $\mathsf{STEP}_i^{q,t}$ as follows:

$$\mathsf{STEP}_i^{q \to q',t} = \begin{cases} x_i^{q',N-1} & q' \in F \\ x_i^{q',N-1} & q' \notin F, q \in Q_i, q' \in Q_j, i \neq j \\ x_i^{q',t-1} & q, q' \in Q_i \text{ for some } i, Q_i \cap F = \emptyset, t > 0 \\ \mathbf{false} & q, q' \in Q_i \text{ for some } i, Q_i \cap F = \emptyset, t = 0 \end{cases}$$

$\mathsf{STEP}_i^{q,t}$ returns the value we need to use on the right-hand side of the implication in Invariance and Progress obligations, wherever x_i^q is used in the original encoding. If q belongs to an accepting set, we use the single copy as before, except that now it is called $x_i^{q,N-1}$ instead of x_i^q. If q and q' belong to the same rejecting set and $t > 0$, then by "moving to" q' we have used up one step, and therefore we will use $x_i^{q,t-1}$. If $t = 0$ then we may not move from q to q' if they are in the same rejecting set. And finally, if q' belongs to a rejecting set but not the one that q belongs to (whether q's set is accepting or rejecting), when we move to q' and enter the new rejecting set we "reset" the counter to $N-1$, to allow N steps in the new set.

The constraints imposed by the transition from q will be denoted $\langle\!\langle \delta(q) \rangle\!\rangle_i^{q,t}$. If $q \in F$, then t will only take the value $N-1$; if $q \notin F$, t will take the values $0, \ldots, N-1$, where $N = D \cdot k$. (Recall that D is the size of the largest rejecting set in A.)

The constraint depends on $\delta(q)$:

$$\langle\!\langle p \rangle\!\rangle_i^{q,t} = \mathsf{L}_p(u_i)$$
$$\langle\!\langle \neg p \rangle\!\rangle_i^{q,t} = \neg \mathsf{L}_p(u_i)$$

$$\langle\!\langle q_1 \wedge q_2 \rangle\!\rangle_i^{q,t} = \mathsf{STEP}_i^{q \to q_1, t} \wedge \mathsf{STEP}_i^{q \to q_2, t}$$
$$\langle\!\langle q_1 \vee q_2 \rangle\!\rangle_i^{q,t} = \mathsf{STEP}_i^{q \to q_1, t} \vee \mathsf{STEP}_i^{q \to q_2, t}$$
$$\langle\!\langle \Diamond q_1 \rangle\!\rangle_i^{q,t} = \bigvee_{j=0}^{k-1} \left(\mathsf{R}(u_i, u_j) \wedge \mathsf{STEP}_j^{q \to q_1, t} \right)$$

The new constraints are used in the new formula as follows.

$$\mathsf{PRF}_{M,A,k}^{\mathsf{weak}} = \mathsf{I}(u_0) \wedge x_0^{q_0, N-1} \wedge \bigwedge_{i=0}^{k-1} \bigwedge_{q \in F} \left(x_i^{q,N-1} \to \langle\!\langle \delta(q) \rangle\!\rangle_i^{q,N-1} \right) \wedge \bigwedge_{i=0}^{k-1} \bigwedge_{q \in Q \setminus F} \bigwedge_{t=0}^{N-1} \left(x_i^{q,t} \to \langle\!\langle \delta(q) \rangle\!\rangle_i^{q,t} \right)$$

Theorem 3.3.1. *For all weak automata A, models M and bounds k, $\mathsf{PRF}_{M,A,k}^{\mathsf{weak}}$ is satisfiable iff there exists a Namjoshi-style proof $\Pi = (I, \sigma)$ with $|\operatorname{domain}(\Pi)| = k$ showing that M satisfies A.*

Proof. If there exists a satisfying assignment v for $\mathsf{PRF}_{M,A,k}^{\mathsf{weak}}$, then we can construct a weak proof $\Pi = (I, \sigma)$ as follows.

- For all $q \in Q$, set $I_q = \left\{ v(u_i) \mid v(x_i^{q,t}) = 1 \text{ for some } 0 \leq t < N \right\}$.
- For all $q \in Q$ and $s \in I_q$, set $\sigma_q(s) = \min \left\{ t \mid v(x_i^{q,t}) = 1 \right\}$.

The proof thus defined is valid:

- Initiality: since $v \models \mathsf{PRF}_{M,A,k}^{\mathsf{weak}}$, in particular $v \models \mathsf{I}(u_0) \wedge x_0^{q_0, N-1}$. Therefore $v(u_0) = s_0$. There exists $t = N - 1$ for which $v(x_0^{q_0,t}) = 1$, so $v(u_0) = s_0 \in I_{q_0}$.
- Consistency: if $s \in I_q$, then there exists an $i \in \{0, \ldots, k-1\}$ such that $s = v(u_i)$ and a $t \in \{0, \ldots, N-1\}$ such that $v(x_i^{q,t}) = 1$. The minimum is well-defined because it is not over an empty set, and hence $\sigma_q(s)$ is defined.
- Invariance and Progress: suppose $s \in I_q$ and $\sigma_q(s) = m$. Then there is an $i \in \{0, \ldots, k-1\}$ and a *minimal* $t \in \{0, \ldots, N-1\}$ such that $s = v(u_i)$ and $v(x_i^{q,t}) = 1$. Since $v \models \mathsf{PRF}_{M,A,k}^{\mathsf{weak}}$, v must satisfy the constraint $x_i^{q,t} \to \langle\!\langle \delta(q) \rangle\!\rangle_i^{q,t}$, and therefore $v \models \langle\!\langle \delta(q) \rangle\!\rangle_i^{q,t}$.

  Also, $\sigma_q(s) = t$, because we chose t to be the minimal value for which $v(x_i^{q,t}) = 1$,

  - If $\delta(q) = p$, then $v \models \mathsf{L}_p(u_i)$, and therefore $p \in L(s)$ and the requirement is satisfied. Similarly if $\delta(q) = \neg p$.
  - If $\delta(q) = q_1 \vee q_2$, then $v \models \mathsf{STEP}_i^{q \to q_1, t} \vee \mathsf{STEP}_i^{q \to q_2, t}$ and therefore either $v \models \mathsf{STEP}_i^{q \to q_1, t}$ or $v \models \mathsf{STEP}_i^{q \to q_2, t}$. Suppose $v \models \mathsf{STEP}_i^{q \to q_1, t}$.

    It cannot be that q, q' belong to the same rejecting set and $t = 0$, because in that case $\mathsf{STEP}_i^{q \to q_1, t} = \mathbf{false}$. Therefore either q, q_1 are in the same rejecting set

but $t > 0$, or q, q_1 are not in the same set, or they are in the same set but it is accepting. In all three cases, since $v \models \mathsf{STEP}_i^{q \to q_1, t}$, there must be some t' such that $v(x_i^{q_1,t'}) = 1$: if q and q_1 belong to the same set and it is a rejecting set, then $t' = t - 1$; if q and q_1 belong to the same set and it is an accepting set, then $t = t' = N - 1$; and if q and q' do not belong to the same set, then again, $t' = N - 1$.

From the definition of I_{q_1} we have that $s \in I_{q_1}$. Also, if q and q' belong to the same rejecting set, then $t' = t-1$, and from the definition, $\sigma_{q_1}(s) = \min\{r \mid v(x_i^{q,r}) = 1\} \leq t' = t - 1 < t = \sigma_q(s)$. If q and q' do not belong to the same set, or they both belong to the same set but it is accepting, then $<_{q \to q_1} = \mathbb{N}^2$. In all cases $\sigma_{q_1}(s) <_{q \to q_1} \sigma_q(s)$, and the Invariance and Progress obligation is satisfied.

The case where $\delta(q) = q_1 \wedge q_2$ is similar.

– If $\delta(q) = \Diamond q_1$, then $v \models \bigvee_{j=0}^{k-1} \left(\mathsf{R}(u_i, u_j) \wedge \mathsf{STEP}_j^{q \to q_1, t}\right)$. Therefore for some $j \in \{0, \ldots, k-1\}$, $(v(u_i), v(u_j)) \in R$ and $v \models \mathsf{STEP}_j^{q \to q_1, t}$. As in the previous case we obtain that $v(u_j) \in I_{q_1}$ and $\sigma_{q_1}(v(u_j)) <_{q \to q_1} \sigma_q(s)$, so the Invariance and Progress obligation is satisfied.

This concludes the proof that the encoding is sound: if $\mathsf{PRF}_{M,A,k}^{\mathsf{weak}}$ is satisfiable, then $M \models A$.

For the other direction, suppose $M \models A$, and let $\Pi = (I, \sigma)$ be a valid proof showing this, with $|\operatorname{domain}(\Pi)| \leq k$. Unfortunately, we cannot use Π as-is to construct a satisfying assignment for $\mathsf{PRF}_{M,A,k}^{\mathsf{weak}}$: first we must "fix" the rank functions so that they only use values between 0 and $N - 1$, where $N = D \cdot k$. We construct a modified proof $\Pi' = (I, \sigma')$ as follows.

Consider each set $Q_i \in P$ separately. If Q_i is an accepting set, then for all $q \in Q_i$ and $s \in I_q$, set $\sigma'_q(s) = N-1$. If Q_i is a rejecting set, let $(s^0, q^0), \ldots, (s^\ell, q^\ell)$ be an ordering of the set $\{(s, q) \mid q \in Q_i \text{ and } s \in I_q\}$, such that for all $0 \leq j < \ell - 1$, $\sigma_{q^j}(s^j) \leq \sigma_{q^{j+1}}(s^{j+1})$. Since there are no more than k model states that participate in the proof, and $|Q_i| \leq D$, we have that $\ell \leq N = D \cdot k$. Now assign ranks as follows: for all $q \in Q$ and $s \in I_q$, let $\sigma_q(s) = j$, where j is the index such that $(s, q) = (s^j, q^j)$.

The new proof assigns only ranks that are no greater than $N-1$. We argue that it is also a valid proof: Initiality and Consistency were not violated by our modification, since we did not change the invariants and we defined a ranks for all $q \in Q$ and $s \in I_q$. Invariance and Progress obligations were also not affected. Take for example $q \in Q$ with $\delta(q) = q_1 \vee q_2$ (the other cases are similar). If $s \in I_q$, then either $s \in I_{q_1}$ and $\sigma_{q_1}(s) <_{q \to q_1} \sigma_q(s)$ or $s \in I_{q_2}$ and $\sigma_{q_2}(s) <_{q \to q_2} \sigma_q(s)$. Suppose it is the first case. If q and q_1 do not belong to the same rejecting set, then regardless of the values assigned to $\sigma'_{q_1}(s)$ and $\sigma'_q(s)$, $\sigma'_{q_1}(s) <_{q \to q_1} \sigma'_q(s)$. If q and q' do belong to the same rejecting set, then in the ordered list of pairs we constructed for that set, the pair (s, q_1) must appear before (s, q), because $\sigma_{q_1}(s) < \sigma_q(s)$. Hence, $\sigma'_{q_1}(s) < \sigma'_q(s)$,

and therefore $\sigma'_{q_1}(s) <_{q \to q_1} \sigma'_q(s)$, and the Invariance and Progress requirement is satisfied.

Using Π' we can construct a satisfying assignment v for $\mathsf{PRF}^{\mathsf{weak}}_{M,A,k}$: assign to each u_i one state from domain(Π') arbitrarily, except for u_0, which must be assigned s_0. For all $i \in \{0, \ldots, k-1\}$, $q \in Q$ and $t \in \{0, \ldots, N-1\}$, set $v(x_i^{q,t}) = 1$ iff $v(u_i) \in I_q$ and $\sigma_q(s) \leq t$. To show that $v \models \mathsf{PRF}^{\mathsf{weak}}_{M,A,k}$, consider the constraints that appear in $\mathsf{PRF}^{\mathsf{weak}}_{M,A,k}$:

- $\mathsf{I}(u_0) \wedge x_0^{q_0,N-1}$: since Π' satisfies Initiality, $s_0 \in I_{q_0}$. Also, $\sigma'_{q_0}(s_0) \leq N-1$, because σ' does not assign ranks greater than $N-1$. Therefore v satisfies this constraint.

- Constraints of the form $x_i^{q,t} \to \langle\!\langle \delta(q) \rangle\!\rangle_i^{q,t}$: let us take for example the case where $\delta(q) = q_1 \vee q_2$. (Again, the other cases are similar.) If $v \models x_i^{q,t}$, then $v(u_i) \in I_q$, and since Π' satisfies Invariance and Progress, either $v(u_i) \in I_{q_1}$ and $\sigma_{q_1}(v(u_i)) <_{q \to q_1} \sigma_q(v(u_i))$ or $v(u_i) \in I_{q_2}$ and $\sigma_{q_2}(v(u_i)) <_{q \to q_2} \sigma_q(v(u_i))$. Suppose it is the first.

  The constraint is of the form

  $$\langle\!\langle q_1 \vee q_2 \rangle\!\rangle_i^{q,t} = \mathsf{STEP}_i^{q \to q_1, t} \vee \mathsf{STEP}_i^{q \to q_2, t}$$

  If q and q_1 do not belong to the same set, or if q is accepting, then $\mathsf{STEP}_i^{q \to q_1, t} = x_i^{q_1, N-1}$. Since $\sigma_{q_1}(v(u_i)) \leq N-1$, $v \models x_i^{q_1,N-1}$, and the constraint is satisfied.

  Suppose q and q_1 belong to the same rejecting set. In that case, $t > 0$, otherwise we would have $\mathsf{STEP}_i^{q \to q_1, t} = \mathbf{false}$, and this cannot be because $v \models \mathsf{STEP}_i^{q \to q_1, t}$. So $t > 0$, and $\mathsf{STEP}_i^{q \to q_1, t} = x_i^{q_1, t-1}$. Since $v \models x_i^{q,t}$ we know that $\sigma'_q(v(u_i)) \leq t$. Also, from the definition of $<_{q \to q_1}$, in this case $\sigma'_{q_1}(v(u_i)) < \sigma'_q(v(u_i))$. Hence, $\sigma'_{q_1}(v(u_i)) \leq t-1$, which means that $v \models x_i^{q_1,t-1}$ (by the definition of v). The constraint is therefore satisfied.

□

As noted in [19], one way to view this encoding is to think of it as summarizing the run of a symbolic model-checker, in particular the Emerson-Lei symbolic model-checking algorithm (Section 2.2.3). In [19], ideas from the world of symbolic model-checking are borrowed and used for linear-time model-checking; here we use them in their original context. Given an automaton obtained by the standard translation from a μ-calculus formula φ, each "row" $x_0^{\langle\psi\rangle,t}, \ldots x_{k-1}^{\langle\psi\rangle,t}$ can be thought of as a characteristic vector for the t-th approximant for ψ, if ψ is a least fixpoint subformula. As in the Emerson-Lei algorith, each approximant is computed by applying the fixpoint body to the previous approximant: in our case this is reflected in the Invariance and Progress constraints, which refer to the $(t-1)$-th "row" $x_0^{\langle\psi\rangle,t-1}, \ldots, x_{k-1}^{\langle\psi\rangle,t-1}$. The analogy breaks down when it comes to greatest fixpoints, because we only use one "row" $x_0^{\psi,N-1}, \ldots, x_{k-1}^{\psi,N-1}$ and always refer to it in the Invariance and Progress obligations. This is more similar to "guessing" an under-approximation A of the greatest fixpoint, and only verifying that it really is an under-approximation by applying the fixpoint body τ_ψ to it and checking that $A \subseteq \tau_\psi(A)$.

3.4 Constructing a Counter-Example

When using bounded model-checking to find bugs, it is useful to be able to present the user with a small counter-example showing the faulty part of the design (if the design does not satisfy the property under testing). Our encoding enables us to do so very easily: given a satisfying assignment v for $\mathsf{PRF}_{M,A,k}$ or one of its variations, the counter-example is the submodel of the design induced by the states $\{v(u_0), \ldots, v(u_{k-1})\}$.

If Alg. 2 is followed, the counter-example is minimal in the number of states it contains; it may, however, contain extraneous edges — transitions that do not actively contribute to the "bad" behavior of the design. If $\mathsf{PRF}^{\mathsf{exp}}_{M,A,k}$, the formula with proof-successors encoded explicitly, is used in the bounded model-checking, we can also eliminate unnecessary edges, and present the user with a counter-example which contains only edges that were used in the proof which shows that it is indeed a counter-example. A transition $(v(u_i), v(u_j)) \in R$ is only *necessary* in the proof represented by the assignment v if for some $q \in Q_\Diamond$, $v(x_i^q) = 1$ and $v(t_i^q) = v(u_j)$. This signifies that $v(u_j)$ is the proof-successor for $v(u_i)$ required by q. If we remove all edges that are not necessary from the counter-example, the proof represented by v is still valid over the submodel we obtain.

In addition to the counter-example itself, we can also return an annotation detailing which counter-example states were found to satisfy which automaton states (or which subformulas, since each automaton state represents a subformula). The annotation for state $v(u_i)$ is simply the collection $\{v(x_i^q) \mid q \in Q\}$. If $v(x_i^q) = 1$, then $v(u_i)$ satisfies q. (However, if $v(x_i^q) = 0$, it cannot be concluded that $v(u_i)$ does not satisfy q; all we know in this case is that we did not need to show that $v(u_i)$ satisfies q to prove that the counter-example satisfies the negation of the formula.)

Remark. Although the counter-example we generate is minimal in the number of states, it is not minimal in the number of edges, even if we remove unnecessary edges. The assignment returned by the SAT solver may not assign proof-successors in an optimal way, as in the following example.

Example 3.4.1. Consider the model $M = (\{s_0, s_1\}, s_0, \delta, L)$ over $AP = \{a, b\}$ shown in Fig. 3.1, with $L(s_0) = \{a\}$ and $L(s_1) = \emptyset$, and with transitions as shown in the figure.

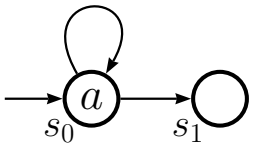

Figure 3.1. The model for Example 3.4.1

The model does not satisfy the μ-calculus property $\varphi = \Box a \vee \Box b$; in other words, it

satisfies $\neg\varphi = \Diamond\neg a \wedge \Diamond\neg b$. The automaton for $\neg\varphi$ is depicted in Fig. 3.2. It is given by $A_{\neg\varphi} = (AP, \{q_0, q_1, q_2, q_3, q_4\}, q_0, \delta, \Omega)$, where Ω is undefined for all the states, and δ is given in Table 3.1.

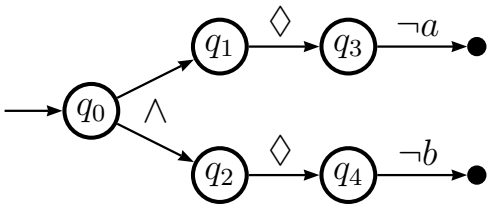

Figure 3.2. The automaton $A_{\neg\varphi}$ from Example 3.4.1

State	Transition	Invariant
q_0	$q_1 \wedge q_2$	$\{s_0\}$
q_1	$\Diamond q_3$	$\{s_0\}$
q_2	$\Diamond q_4$	$\{s_0\}$
q_3	$\neg a$	$\{s_1\}$
q_4	$\neg b$	$\{s_0, s_1\}$

Table 3.1. The transitions of $A_{\neg\varphi}$ and the invariants of Π from Example 3.4.1

A possible proof for $M \models \neg\varphi$ is $\Pi = (I, \rho)$, where $\rho_q(s) = 0$ for all automaton states q and model states s (since there are no fixpoints, ranks are not needed), and the invariants are given in Table 3.1. The proof is minimal in the number of model states it contains — s_0 is needed because it is the initial state, and s_1 is needed as witness that $s_0 \models \Diamond\neg a$. The edge (s_0, s_1) must appear in the counter-example. However, the edge (s_0, s_0) may or may not appear, depending on the assignment returned by the SAT solver: one possibility is to regard s_1 as the proof-successor for s_0 required by q_2, in which case the edge is not necessary; but another possibility is to regard s_0 as its own proof-successor, in which case the edge (s_0, s_0) becomes necessary. The counter-example resulting from the first possibility is minimal in the number of edges; the counter-example obtained from the second possibility is not, but it is an equally valid possibility.

Chapter 4

Completeness Thresholds

The encodings presented in the previous section provide a way to determine when a Kripke structure does *not* satisfy a $\square$-automaton A or $\square L_\mu$ formula φ: construct the complement $A_\neg$ of A or φ, choose a bound k, and if a SAT solver returns a satisfying assignment for $\mathsf{PRF}_{M,A_\neg,k}$ then M does not satisfy A or φ. However, Alg. 2 can only conclude that M *does* satisfy A if $\mathsf{PRF}_{M,A_\neg,k}$ is not satisfiable when k is the number of states in M. This is a very pessimistic bound; in practice, it is likely that the formula $\mathsf{PRF}_{M,A_\neg,k}$ will become intractable for the SAT solver long before k reaches $|S|$. We are interested in developing better ways to determine when, upon finding that $\mathsf{PRF}_{M,A_\neg,k}$ is unsatisfiable for some $k \leq |S|$, it is safe to conclude that M satisfies A or φ.

4.1 A Static Completeness Threshold

A *static completeness threshold* is a conservative bound $\mathrm{CT}(M, \varphi)$ such that when bounded model-checking with a bound of $k = \mathrm{CT}(M, \varphi)$ fails to find a counter-example, it can soundly be concluded that $M \models \varphi$. In this section we present a static completeness threshold for ECTL.

Given a model $M = (S, s_0, R, L)$, the *recurrence diameter* of M, denoted r_M, is the smallest natural number such that for every (not necessarily initialized) path $\pi = t_0 \dots t_{r_M+1}$ there exist $i, j \leq r_M + 1$ for which $t_i = t_j$. In words, r_M is the length of the longest simple path in M (where the length of a path is considered to be the number of states on the path, not edges). In particular, if π is an infinite path, then among the first $r_M + 1$ states of π there must be a state that appears twice.

We define a static completeness threshold for an ECTL formula φ by induction on the structure of φ:

- $\mathrm{CT}(M, \mathbf{true}) = 1$.
- For $p \in AP$, $\mathrm{CT}(M, p) = \mathrm{CT}(M, \neg p) = 1$.

- $\mathrm{CT}(M, \varphi_1 \vee \varphi_2) = \max(\mathrm{CT}(M, \varphi_1), \mathrm{CT}(M, \varphi_2))$.
- $\mathrm{CT}(M, \varphi_1 \wedge \varphi_2) = \mathrm{CT}(M, \varphi_1) + \mathrm{CT}(M, \varphi_2) - 1$.
- $\mathrm{CT}(M, EX\varphi_1) = 1 + \mathrm{CT}(M, \varphi_1)$.
- $\mathrm{CT}(M, E[\varphi_1 \, U \, \varphi_2]) = (r_M - 1) \cdot \mathrm{CT}(M, \varphi_1) + \mathrm{CT}(M, \varphi_2)$.
- $\mathrm{CT}(M, EG\varphi_1) = r_M \cdot \mathrm{CT}(M, \varphi_1)$.

For $\varphi = EF\varphi_1$, we derive from the equivalence $EF\varphi_1 \equiv E[\mathbf{true} \, U \, \varphi_1]$ a completeness threshold of $\mathrm{CT}(M, \varphi) = (r_M - 1) \cdot \mathrm{CT}(M, \mathbf{true}) + \mathrm{CT}(M, \varphi_1) = r_M + \mathrm{CT}(\varphi_1) - 1$.

Theorem 4.1.1. *For any model M, state s of M and ECTL property φ, if $M, s \models \varphi$ then there exists a submodel N of M with at most $\mathrm{CT}(M, \varphi)$ states, such that s is the initial state of N and $N, s \models \varphi$.*

Proof. The proof is by induction on φ. Let $M = (S, s_0, R, L)$ be a model.

For the base case, if $\varphi = p$ for some $p \in AP$ and $M, s \models \varphi$, then $p \in L(s)$; a submodel N of size $\mathrm{CT}(p) = 1$ is simply given by $N = (\{s\}, s, \emptyset, L|_{\{s\}})$. For $\neg p$ the proof is similar.

Suppose the claim holds for φ_1 and for φ_2. That is, for all states s, if $M, s \models \varphi_1$ then there exists a submodel N of M with at most $\mathrm{CT}(M, \varphi_1)$ states and with s as its initial state, such that $N, s \models \varphi_1$, and similarly for φ_2.

For $\varphi = \varphi_1 \vee \varphi_2$: if $M, s \models \varphi_1 \vee \varphi_2$, then either $M, s \models \varphi_1$ or $M, s \models \varphi_2$. In the first case, from the induction hypothesis there exists a submodel N_1 of M with $\mathrm{CT}(M, \varphi_1)$ states, with s as its initial state, such that $N_1, s \models \varphi_1$. In the second case, there exists a submodel N_2 of M with $\mathrm{CT}(M, \varphi_2)$ states, with s as its initial state, such that $N_2, s \models \varphi_2$. Either way, there exists a submodel of at most $\max(\mathrm{CT}(M, \varphi_1), \mathrm{CT}(M, \varphi_2))$ with s as its initial state, and which satisfies $\varphi_1 \vee \varphi_2$.

For $\varphi = \varphi_1 \wedge \varphi_2$: If $M, s \models \varphi_1 \wedge \varphi_2$, then $M, s \models \varphi_1$ and $M, s \models \varphi_2$. From the induction hypothesis, there exist submodels $N_1 = (S_1, s, R_1, L|_{S_1})$ and $N_2 = (S_2, s, R_2, L|_{S_2})$ comprising at most $\mathrm{CT}(M, \varphi_1)$ and $\mathrm{CT}(M, \varphi_2)$ states, respectively, such that $N_1, s \models \varphi_1$ and $N_2, s \models \varphi_2$. Construct a submodel $N = (S_1 \cup S_2, s, R_1 \cup R_2, L|_{S_1 \cup S_2})$. Since N_1 and N_2 share the state s, $|S_1 \cup S_2| \leq |S_1| + |S_2| - 1 \leq \mathrm{CT}(M, \varphi_1) + \mathrm{CT}(M, \varphi_2) - 1$. In addition, N_1 and N_2 are both submodels of N with the same initial state s, and therefore s satisfies in N all the ECTL properties that s satisfies in N_1 or in N_2. In particular, $N, s \models \varphi_1$ and $N, s \models \varphi_2$, and therefore $N, s \models \varphi_1 \wedge \varphi_2$.

For $\varphi = EX\varphi_1$: if $M, s \models EX\varphi_1$ then there exists $s' \in S$ such that $(s, s') \in R$ and $M, s' \models \varphi_1$. From the induction hypothesis, there exists a submodel $N = (S', s', R', L|_{S'})$ with at most $\mathrm{CT}(M, \varphi_1)$ states such that $N, s' \models \varphi_1$. Construct a submodel $N_X = (S' \cup \{s\}, s, R' \cup \{(s, s')\}, L|_{S' \cup \{s\}})$. N_X has at most $1 + \mathrm{CT}(M, \varphi_1)$ states, its initial state is s, and it satisfies $N, s \models EX\varphi_1$.

For $\varphi = EG\varphi_1$: if $M, s \models EG\varphi_1$ then there exists a path $\pi = t_0 t_1 \ldots$ in M such that $t_0 = s$ and for all $i \in \mathbb{N}$, $M, t_i \models \varphi_1$. π is an infinite path, and therefore there exist $i, j \leq r_M + 1$ such that $t_i = t_j$. Consider the path $\pi' = t_0 \ldots t_{i-1}(t_i \ldots t_{j-1})^\omega$. π' is a path of M, because $t_i = t_j$; since π is a path of M, $(t_{j-1}, t_j) = (t_{j-1}, t_i) \in R$, and also for all $\ell \neq j-1$, $(t_\ell, t_{\ell+1}) \in R$. The number of distinct states that appear in π' is no greater than $j-1$, which is at most r_M. From the induction hypothesis, for all $\ell \leq j-1$ there exists a submodel $N_\ell = (S_\ell, t_\ell, R_\ell, L|_{S_\ell})$, containing at most $\mathrm{CT}(M, \varphi_1)$ states, such that $N_\ell, t_\ell \models \varphi_1$.

Let $S' = \bigcup_{\ell=0}^{j-1} S_\ell$, and construct a submodel $N = (S', t_0, R|_{S'}, L_{S'})$. Since each submodel N_ℓ contains t_ℓ as its initial state, the resulting submodel N contains the path π'; and since each N_ℓ is a submodel of N, $N, t_\ell \models \varphi_1$ for all $\ell \in \{0, \ldots, j-1\}$. Therefore $N, t_0 \models EG\varphi_1$. The initial state is $t_0 = s$, and the number of states in N is at most $r_M \cdot \mathrm{CT}(M, \varphi_1)$.

For $\varphi = E[\varphi_1 U \varphi_2]$ the proof is similar to the case of EG. If $M, s \models E[\varphi_1 U \varphi_2]$, then there exists a path $\pi = t_0 t_1 \ldots$ in M, such that for some $i \geq 0$, $M, t_i \models \varphi_2$, and for all $j < i$, $M, t_j \models \varphi_1$. If $i > r_M$, then there is a loop in the prefix $t_0 \ldots t_i$, and we can remove it and obtain a shorter path that still satisfies $\varphi_1 U \varphi_2$. We can continue to remove loops in the prefix until we have obtained a path $\pi' = t'_0 t'_1 \ldots$ such that there exists $i' \leq r_M$ for which $M, t_{i'} \models \varphi_2$ and for all $j < i'$, $M, t_j \models \varphi_1$. From the induction hypothesis, for all $j < i'$ there exists a submodel $N^1_j = (S^1_j, t_j, R|_{S^1_j}, L|_{S^1_j})$ with at most $\mathrm{CT}(M, \varphi_1)$ states, such that $N^1_j, t_j \models \varphi_1$. There also exists a submodel $N^2 = (S^2, t_{i'}, R|_{S^2}, L|_{S^2})$ with at most $\mathrm{CT}(M, \varphi_2)$ states, such that $N^2, t_{i'} \models \varphi_2$.

Let $S' = S^2 \cup \bigcup_{j=0}^{i'-1} S^1_j$, and construct a submodel $N = (S', t_0, R|_{S'}, L_{S'})$. Since $i' \leq r_M$, the number of states in N is at most $(r_M - 1) \cdot \mathrm{CT}(M, \varphi_1) + \mathrm{CT}(M, \varphi_2)$. The path π' is a path of N, and N^2 and each N^1_j are submodels of N. Therefore, similar to the case of EG, $N, t_0 \models E[\varphi_1 U \varphi_2]$. □

As a consequence of Theorem 4.1.1, Alg. 2 can now be re-written as follows.

Algorithm 3 Bounded Model-Checking Using the Static Completeness Threshold

```
function bmc(M, φ)
A ← to-automaton(¬φ)
for k = 1 to CT(M, φ) do
  res ← SAT-solve(PRF_{M,A,k})
  if res = sat then
    return false
  end if
end for
return true
```

4.2 A Dynamic Completeness Criterion

The static completeness criterion presented in the previous section, although a better bound than $|S|$, is still not of much practical use. Computing the recurrence diameter r_M is difficult in practice. It is possible to compute the recurrence diameter *dynamically* [4], by testing the satisfiability of the following formula at each value of k:

$$\bigwedge_{0 \le i < k-1} \mathsf{R}(u_i, u_{i+1}) \wedge \bigwedge_{0 \le i \ne j \le k-1} (u_i \neq u_j)$$

When the formula becomes unsatisfiable for the first time, we know that k has reached the recurrence diameter: there exists no path of length k all of whose states are distinct — in other words, every path of length k contains a state that appears twice on the path. However, even if we compute the recurrence diameter dynamically and use the bound computed in the previous section, we would still have a rather pessimistic bound, which does not take into account the finer details of the formula and model.

Following [42], we are interested in a *dynamic completeness criterion*: a formula $\mathsf{CMP}_{M,A,k}$ that is satisfiable while there is still hope of finding a counter-example, and that becomes unsatisfiable when there is none. In the bounded model-checking algorithm, after failing to find a counter-example of size k (that is, when $\mathsf{PRF}_{M,A,k}$ is unsatisfiable), we will check whether $\mathsf{CMP}_{M,A,k}$ is satisfiable; if it is not we will conclude that $M \not\models A$. If it is satisfiable, we will increase the bound k and go to the next iteration.

In [42], a dynamic completeness criterion is developed for the tree-based encoding. (The construction is presented in Section 5.2.) Here we develop a dynamic completeness criterion for our graph-based encoding. The two constructions are very different from each other, because of the differences between the two encodings. The underlying idea in both cases is to identify situations where a sufficiently large fragment of the model has already been explored, and exploring a larger fragment (that is, using a larger bound) will not lead to the discovery of a counter-example. However, in the encoding of [42] the shape of the counter-example is known in advance, which makes it easier to construct a dynamic completeness criterion. In developing a completeness criterion for our encoding, we must contend with the fact that we do not know in advance how the k counter-example states will be arranged.

Essentially, $\mathsf{CMP}_{M,A,k}$ should encode the fact that it is possible to arrange k states so that they form a "beginning" of a proof, which might be extended into a valid proof by adding more states. In the remainder of this section we will formally define what a "beginning" of a proof is, and show how to construct a Boolean formula which is satisfiable iff there exists a beginning of a proof containing k states.

Example 4.2.1. As an example for the inadequacy of the static completeness criterion, consider the formula $\varphi = E[p\, U\, q]$, and the model M shown in Fig. 4.1. The model has a

recurrence diameter of 4 — the longest simple path in M is $s_0 s_1 s_3 s_2$, comprising four states.

The automaton that corresponds to φ is shown in Fig. 4.2; the priority of all the states is undefined, except for q_0, which has a priority of 1, reflecting the fact that Until is a liveness property and after a finite number of steps q has to be satisfied. The transitions of the automaton, and the subformulas to which each state corresponds, are shown in Table 4.1. (The subformulas are written in ECTL to simplify the presentation; the automaton itself is constructed from the μ-calculus equivalent of *phi*.)

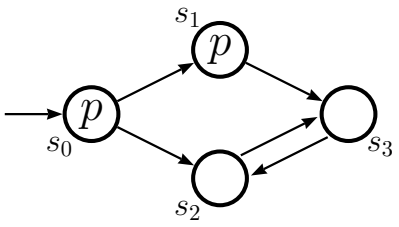

Figure 4.1. The model from Example 4.2.1

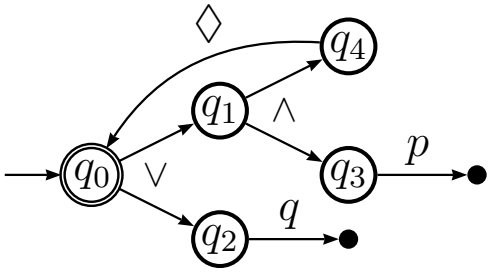

Figure 4.2. The automaton for $E[p\,U\,q]$ from Example 4.2.1

State	Transition	Subformula(s)	Priority
q_0	$q_1 \vee q_2$	$E[p\,U\,q] \equiv p \vee (q \wedge EXE[p\,U\,q])$	1
q_1	$q_3 \wedge q_4$	$q \wedge EXE[p\,U\,q]$	undefined
q_2	q	q	undefined
q_3	p	p	undefined
q_4	$\Diamond q_0$	$EXE[p\,U\,q]$	undefined

Table 4.1. The transitions of the automaton for $E[p\,U\,q]$ from Example 4.2.1

The static completeness threshold for φ is $r_M = 4$. A witness for $E[p\,U\,q]$ is a finite path that contains only states labeled with p, except possibly the last state, which should satisfy q. In M, the length of the longest path that contains only states labeled with p is 2. Therefore, with a bound of 3, we cannot find a *beginning* of a counter-example — a structure that might be extended into a counter-example by adding more states. When k reaches 3, we would like to be able to determine that $M \not\models E[p\,U\,q]$.

Formally, we say that $\Upsilon = (I, \rho)$ is a *partial proof* if it satisfies the following conditions.

1. Initiality: $s_0 \in I_{q_0}$.

2. Partial Invariance and Progress: For all $q \in Q \setminus Q_\Diamond$, *Strong* Invariance and Progress obligations are imposed. These are the usual Invariance and Progress obligations: for all $s \in I_q$,

   - If $\delta(q) = p$ then $p \in L(s)$.
   - If $\delta(q) = \neg p$ then $p \notin L(s)$.
   - If $\delta(q) = q_1 \vee q_2$, then either $s \in I_{q_1}$ and $\rho_{q_1}(s) \circledcirc_q \rho_q(s)$, or $s \in I_{q_2}$ and $\rho_{q_2}(s) \circledcirc_q \rho_q(s)$.
   - If $\delta(q) = q_1 \wedge q_2$, then $s \in I_{q_1}$ and $\rho_{q_1}(s) \circledcirc_q \rho_q(s)$, and also $s \in I_{q_2}$ and $\rho_{q_2}(s) \circledcirc_q \rho_q(s)$.

   For all $q \in Q_\Diamond$, *Weak* Invariance and Progress is required. If $\delta(q) = \Diamond q_1$, for all $s \in I_q$, at least one of the following must hold:

   - Regular Invariance and Progress: there exists $t \in S$ such that $(s, t) \in R$ and $t \in I_{q_1}$ and $\rho_{q_1}(t) \circledcirc_q \rho_q(s)$; or,
   - "Escape clause": there exists a fresh state $t \notin \text{domain}(\Upsilon)$ such that $(s, t) \in R$. (Recall that $\text{domain}(\Upsilon) = \bigcup_{q \in Q} I_q$ is the set of states that participate in Υ.)

   If the first condition holds, we say that the Invariance and Progress obligation for q and s is *satisfied*. If the first condition does not hold, and the "escape clause" is satisfied, we say that the obligation is *violated*. (It is possible for both conditions to hold at the same time, in which case we say that the obligation is satisfied.)

This definition formalizes what it means for a structure to be a "beginning" of a proof, but it is not sufficient for our purposes. We would like to be able to terminate the BMC procedure at the smallest possible bound. We are hindered by the fact that we can often add states to a partial proof without invalidating it. Thus, we may think that there exists a "beginning" of a proof comprising k states, when more careful analysis would reveal that the "beginning" we found contains unnecessary states.

For instance, in Example 4.2.1, one partial proof for $M \models \varphi$ is given by $\Upsilon = (I, \rho)$ where $I_{q_0} = I_{q_1} = I_{q_3} = \{s_0, s_1\}$, $I_{q_4} = \{s_0, s_1, s_2\}$ and $I_{q_2} = \emptyset$. The rank assignment is immaterial. Υ is a partial proof: s_0 satisfies all its obligations; s_1, s_2 violate the Invariance and Progress obligation for q_4, but both have a transition to s_3, which does not participate in Υ. It may therefore seem like there is hope for extending Υ into a valid proof by adding s_3. But we have already seen that there is no "beginning of a proof" with 3 states, so Υ must somehow not comply with our intention.

Indeed, we can see that s_2 has been added to Υ spuriously. It does not serve any useful function in the partial proof, since removing it would not affect the satisfaction of any

obligations, and it inflates the number of states that participate in Υ. We would like to avoid having "useless" states like s_2 in the partial proof in order to be able to halt the bounded model-checking with a smaller bound. We will now refine the definition to prevent the participation of useless states.

Given a proof $\Pi = (I, \rho)$, we define a *proof-successor assignment* PS as a collection $PS = \{PS_q \mid q \in Q_\Diamond\}$, where $PS_q : I_q \to \text{domain}(\Pi)$ assigns to each $s \in I_q$ a proof-successor of s for q; that is, if $\delta(q) = \Diamond q_1$, then $PS_q(s)$ satisfies $PS_q(s) \in I_{q_1}$ and $\rho_{q_1}(PS_q(s)) \circledcirc_q \rho_q(s)$. There may exist more than one possible proof-successor assignment for a given proof, as can be seen in Example 3.4.1, where different proof-successor assignments yield counter-examples with different numbers of edges.

With regard to a proof-successor assignment PS, define a graph $G_{\Pi,PS} = (\text{domain}(\Pi), E)$ where $E = \{(s, PS_q(s)) | q \in Q_\Diamond \text{ and } s \in I_q\}$. Each edge in E will be called a *dependency*.

Define the set $\text{NEC}_{\Pi,PS}$ of *necessary states* w.r.t. PS in Π to be the states reachable from s_0 in $G_{\Pi,PS}$. In addition, define a relation $\text{REQ}_{\Pi,PS} \subseteq E$ by: $(s,t) \in \text{REQ}_{\Pi,PS}$ iff s immediately precedes t on a shortest path from s_0 to t; that is, there exists a shortest path $\pi = s_0 \dots s_{\ell-1} s_\ell$ in $G_{\Pi,PS}$ such that $s_{\ell-1} = s$ and $s_\ell = t$. The edges in $\text{REQ}_{\Pi,PS}$ are the edges that would be explored in a BFS traversal of $G_{\Pi,PS}$ starting from s_0. The relation $\text{REQ}_{\Pi,PS}$ will be used in our proof of the soundness of the dynamic completeness criterion.

We say that a proof is *PS-concise* if all its states are necessary with regard to PS, and we say that a proof is *concise* if it is PS-concise for *all* possible choices of PS. If there is no proof-successor assignment PS such that the proof is PS-concise, we say that the proof is *wasteful*.

Example 4.2.2. Consider the automaton $A_{\neg\varphi}$ from Example 3.4.1, and the model M shown in Fig. 4.3. The automaton $A_{\neg\varphi}$ is shown in Fig. 4.4. In Table 4.2 we show the transitions of the automaton, and the invariants for three possible proofs showing that $M \models A_{\neg\varphi}$. The ranks are immaterial, since the priority of all automaton states is undefined.

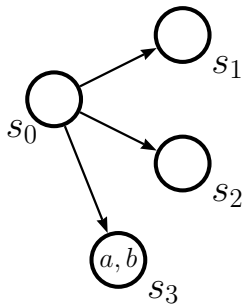

Figure 4.3. The model from Example 4.2.2

The only state that requires a proof-successor in Π_1, Π_2 is s_0, because $q_1, q_2 \in Q_\Diamond$ and s_0 belongs to the invariants I_{q_1} and I_{q_2}. For Π_1 there are four possible proof-successor

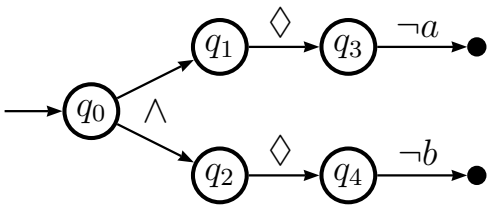

Figure 4.4. The automaton $A_{\neg\varphi}$ from Example 3.4.1 and Example 4.2.2

State	Subformula	Transition	Π_1 Invariant	Π_2 Invariant	Π_3 Invariant
q_0	$EX\neg a \wedge EX\neg b$	$q_1 \wedge q_2$	$\{s_0\}$	$\{s_0\}$	$\{s_0\}$
q_1	$EX\neg a$	$\Diamond q_3$	$\{s_0\}$	$\{s_0\}$	$\{s_0\}$
q_2	$EX\neg b$	$\Diamond q_4$	$\{s_0\}$	$\{s_0\}$	$\{s_0\}$
q_3	$\neg a$	$\neg a$	$\{s_1, s_2\}$	$\{s_1\}$	$\{s_1\}$
q_4	$\neg b$	$\neg b$	$\{s_1, s_2\}$	$\{s_1\}$	$\{s_1, s_3\}$

Table 4.2. The transitions of $A_{\neg\varphi}$ and the invariants of Π_1, Π_2 and Π_3 from Example 3.4.1

assignments: as the proof-successor required by q_1, we can choose either s_1 or s_2, and similarly for the proof-successor required by q_2. For Π_2 and Π_3 there is only one possible proof-successor assignment PS, with $PS_{q_1}(s_0) = PS_{q_2}(s_0) = \{s_1\}$. ($s_2$ and s_3 cannot be chosen, because they are not in I_{q_3} and I_{q_4}.)

Let us denote by PS^1 the proof-successor assignment for Π_1 that has $PS^1_{q_1}(s_0) = s_1$ and $PS^1_{q_2}(s_0) = s_2$, and denote by PS^2 the proof-successor assignment for Π_1 that has $PS^2_{q_1}(s_0) = PS^2_{q_2}(s_0) = s_1$.

Π_1 is PS^1-concise: s_0 is always necessary, being the initial state; s_1 is necessary because $PS^1_{q_1}(s_0) = s_1$, and s_2 is necessary because $PS^1_{q_2}(s_0) = s_2$. However, Π_1 is not PS^2-concise: w.r.t. PS^2, the state s_2 is not necessary, because it has no incoming dependency edges. Since there exists a proof-successor assignment PS^2 such that Π_1 is not PS^2-concise, Π_1 is not concise.

Now consider Π_2. It is concise with regard to the only proof-successor assignment it has: in PS (defined above), s_0 is necessary because it is initial, and s_1 is necessary because $PS_{q_1}(s_0) = s_1$. Therefore, Π_2 is concise.

In contrast, Π_3 is not PS-concise: s_3 is not necessary. Since PS is the only proof-successor assignment for Π_3, and Π_3 is not PS-concise, Π_3 is wasteful.

From the example we can see that there is a "hierarchy of conciseness" among proofs: at the highest level of the hierarchy we have concise proofs (like Π_2). Then we have non-concise proofs which are PS-concise w.r.t. *some* proof-successor assignment PS (like Π_1, which is PS^1-concise but not PS^2-concise). Finally, at the lowest level, we have wasteful proofs (like

Π_3), proofs that are not PS-concise w.r.t. *all* proof-successor assignments PS.

Concise proofs are better than non-concise proofs, because all the states that participate in a concise proof are strictly necessary. A non-concise-proof, in contrast, contains at least one state that can be eliminated without invalidating the proof. (In our example, s_2 can be removed from Π_1.) However, a non-concise but non-wasteful proof like Π_1 is still better in our eyes than a wasteful proof like Π_3. This is because when we take a full proof and "chop off" parts of it to form a partial proof, we obtain a partial proof that is not necessarily concise, but certainly not wasteful. We phrase this more formally in Theorem 4.2.1.

Next, we show that there always exists a concise proof if there exists a proof at all. Given a proof $\Pi = (I, \rho)$ and a set $A \subseteq \text{domain}(\Pi)$, we define the proof Π *restricted to* A, denoted $\Pi|_A$, by $\Pi|_A = (I', \rho')$, where for all $q \in Q$, $I'_q = I_q \cap A$ and $\rho'_q = \rho_q|_A$.

Lemma 4.2.1. *If Π is a valid proof and PS is a proof-successor assignment, then $\Pi|_{\text{NEC}_{\Pi,PS}}$ is also a valid proof.*

Proof. The new proof $\Pi|_{\text{NEC}_{\Pi,PS}}$ satisfies all the proof obligations: suppose $\Pi = (I, \rho)$ and $\Pi|_{\text{NEC}_{\Pi,PS}} = (I', \rho')$.

1. Initiality: s_0 is always reachable from itself, and so $s_0 \in \text{NEC}_{\Pi,PS}$ for all proof-successor assignments PS. Since Π itself is valid, $s_0 \in I_{q_0}$, and hence, $s_0 \in I_{q_0} \cap \text{NEC}_{\Pi,PS}$.

2. Consistency: if $s \in I'_q$, then $s \in I_q$ and $s \in \text{NEC}_{\Pi,PS}$. Since Π satisfies Consistency, $\rho_q(s)$ is defined, and therefore $\rho_q|_{\text{NEC}_{\Pi,PS}}(s)$ is also defined.

3. Invariance and Progress: for all states $q \in Q \backslash Q_\Diamond$, Invariance and Progress requirements are satisfied because each model state s is either removed from all invariants, or it remains in all the invariants to which it belongs in Π and also maintains its ranks. Since Π satisfies Invariance and Progress, and Invariance and Progress obligations for $q \in Q \setminus Q_\Diamond$ involve only a single state, $\Pi|_{\text{NEC}_{\Pi,PS}}$ satisfies these obligations for $q \in Q \setminus Q_\Diamond$.

   For $q \in Q_\Diamond$ and $s \in I'_q$, $s \in I_q$ as well (because $I'_q \subseteq I_q$). Hence, $PS_q(s)$ is defined, and it assigns to s one of its proof-successors t. t is a necessary state: s itself is necessary, otherwise it would not be in I'_q; hence it is reachable from s_0 via dependencies of the form $(s', PS_{q'}(s'))$. t is reached from s by the edge $(s, PS_q(s))$, and it is therefore also reachable from s_0 via dependency edges, which means it is necessary w.r.t. PS. And since $t \in \text{NEC}_{\Pi,PS}$, t belongs to the same invariants in $\Pi|_{\text{NEC}_{\Pi,PS}}$ as it does in Π and also has the same ranks as it did in Π, as does s. Thus, t serves as a proof-successor for s in $\Pi|_{\text{NEC}_{\Pi,PS}}$, and the Invariance and Progress obligation is satisfied.

□

Lemma 4.2.2. *Given a model M and automaton A, M satisfies A iff there is a concise and valid proof Π showing that M satisfies A.*

Proof. Since a concise valid proof is in particular a valid proof, the soundess of Namjoshi's proof system guarantees that the existence of a concise valid proof implies $M \models A$.

In the other direction, suppose $M \models A$, and let Π be a valid proof showing this. If Π is concise, we are done. Otherwise, there exists a proof-successor assignment PS with regard to which $\mathrm{NEC}_{\Pi,PS} \subsetneq \mathrm{domain}(\Pi)$. Construct a series of proofs $\Pi_0, \Pi_1, \ldots, \Pi_\ell$ inductively by $\Pi_0 = \Pi$ and $\Pi_{i+1} = \Pi_i|_{\mathrm{NEC}_{\Pi_i,PS_i}}$ while there remains a proof-successor assignment PS_i such that Π_i is not PS_i-concise. (If there is more than one possible proof-successor assignment PS_i, choose one arbitrarily.) The series is finite, because at each step we remove at least one (unnecessary) state from $\mathrm{domain}(\Pi_i)$; eventually we will run out of states to remove. The final proof Π_ℓ is concise, because there exist no proof-successor assignments PS with regard to which it is not PS-concise. Also, it is a valid proof: by Lemma 4.2.1, the validity of the proof is preserved at each step, and since we started with a valid proof Π, we will end up with a valid proof Π_ℓ. We have therefore shown a concise and valid proof Π_ℓ showing that M satisfies A. □

We extend the definition of proof-successor assignments to partial proofs as follows. If $\Upsilon = (I, \rho)$ is a partial proof, then a proof-successor assignment $PS = \{PS_q \mid q \in Q\}$ for Υ is defined as for a full proof, with the exception that $PS_q(s)$ may not be defined even if $s \in I_q$. (That is, PS_q is a partial function from I_q instead of a full function.) We define PS-conciseness, conciseness and wastefulness for partial proofs the same way as for full proofs.

Before we turn to the construction of the formula $\mathsf{CMP}_{M,A,k}$, we need one more technical lemma.

Lemma 4.2.3. *The graph $G = G(\mathrm{NEC}_{\Pi,PS}, \mathrm{REQ}_{\Pi,PS})$ is a directed acyclic graph (DAG).*

Proof. Suppose by way of contradiction that G contains a cycle $\pi = t_0 t_1 \ldots t_\ell$ where $\ell > 0$, $t_0 = t_\ell$ and for all $i \in \{0, \ldots, \ell - 1\}$, $(t_i, t_{i+1}) \in \mathrm{REQ}_{\Pi,PS}$. Let d be the shortest-path distance of t_0 from s_0. (d is defined, because t_0 is necessary and hence it is reachable from s_0 in G.) From the definition of $\mathrm{REQ}_{\Pi,PS}$, each edge (t_i, t_{i+1}) lies on a shortest path from s_0 to t_{i+1}. It is easy to show by induction that the shortest-path distance of t_i from s_0 is $d + i$. Consequently, we have that the shortest-path distance from s_0 to $t_0 = t_\ell$ is both d and $d + \ell$, which is a contradiction since $\ell > 0$. □

Now we are ready to construct $\mathsf{CMP}_{M,A,k}$. Recall that $\mathsf{CMP}_{M,A,k}$ is intended to encode the existence of a "beginning" of a proof for $M \models A$, with k states. We can now phrase our intention more formally: $\mathsf{CMP}_{M,A,k}$ should be satisfiable iff there exists a partial proof Π with

$|\operatorname{domain}(\Pi)| = k$, which non-wasteful (that is, it is PS-concise w.r.t. some proof-successor assignment PS).

Recall that in the explicit-successor encoding (defined in Section 3.2.2), a satisfying assignment also tells us which states serve as proof-successors for which other states. We need this information now, and therefore the formula $\mathsf{CMP}_{M,A,k}$ will be based on $\mathsf{PRF}^{\mathsf{exp}}_{M,A,k}$. To express the weakened Invariance and Obligation requirements of a partial proof, we define weakened constraints $\langle\!\langle \delta(q) \rangle\!\rangle_i^{\mathsf{partial}}$ as follows:

- For all $q \in Q \setminus Q_\Diamond$, $\langle\!\langle \delta(q) \rangle\!\rangle_i^{\mathsf{partial}} = \langle\!\langle \delta(q) \rangle\!\rangle_i^{\mathsf{exp}}$.
- For $q \in Q_\Diamond$ with $\delta(q) = \Diamond q_1$,

$$\langle\!\langle \Diamond q_1 \rangle\!\rangle_i^{\mathsf{partial}} = \mathsf{R}(u_i, t_i^q) \wedge \bigwedge_{j=0}^{k-1} \left((t_i^q = u_j) \rightarrow \left(x_j^{q_1} \wedge \mathsf{P}_q(\rho_j^{q_1}, \rho_i^q) \right) \right)$$

The new constraints allow for violated Invariance and Progress obligations for $Q_\Diamond$-states, but only if the proof successor is not one of the proof states $u_0, \ldots, u_{k-1}$.

The formula $\mathsf{CMP}_{M,A,k}$ is composed of the following constraints.

- Partial proof:

$$\mathsf{PPRF}_{M,A,k} = \mathsf{I}(u_0) \wedge x_0^{q_0} \wedge \bigwedge_{i=0}^{k-1} \bigwedge_{q \in Q} x_i^q \rightarrow \langle\!\langle \delta(q) \rangle\!\rangle_i^{\mathsf{partial}}$$

- Conciseness:

$$\mathsf{CON}_{A,k} = \bigwedge_{i=1}^{k-1} \left(\bigvee_{j=0}^{i-1} \bigvee_{q \in Q_\Diamond} (u_i = t_j^q) \wedge x_j^q \right)$$

  This constraint enforces the non-wastefulness of the partial proof, by requiring that each participating state except s_0 have an incoming dependency from a preceding proof state.

- Distinctness:

$$\mathsf{D}_k = \bigwedge_{0 \le i \le k-1} \left(\bigwedge_{i < j \le k-1} u_i \neq u_j \right)$$

To obtain $\mathsf{CMP}_{M,A,k}$, we take the conjunction of all the constraints defined above:

$$\mathsf{CMP}_{M,A,k} = \mathsf{PPRF}_{M,A,k} \wedge \mathsf{CON}_{A,k} \wedge \mathsf{D}_k$$

Remark. The distinctness constraint is added to prevent situations where the SAT solver assigns the same model state to different state vectors, inflating the number of states that appear to have been used and preventing termination when it may otherwise be possible.

Theorem 4.2.1 (Soundness)**.** *If* $\mathsf{PRF}_{M,A,k}$ *is unsatisfiable and* M *satisfies* A*, then* $\mathsf{CMP}_{M,A,k}$ *is satisfiable.*

Proof. If M satisfies A then from Lemma 4.2.2 there exists a valid and concise proof Π witnessing this fact. Π cannot have k states or less, otherwise $\mathsf{PRF}_{M,A,k}$ would be satisfiable.

Let PS be a proof-successor assignment for Π. As shown in Lemma 4.2.3, the graph $G(\mathrm{NEC}_{\Pi,PS}, \mathrm{REQ}_{\Pi,PS})$ is a DAG, and therefore there exists a topological sort of its vertices. In our case, since Π is concise, $\mathrm{NEC}_{\Pi,PS} = \mathrm{domain}(\Pi)$, and we can sort all the states that participate in Π topologically with regard to the $\mathrm{REQ}_{\Pi,PS}$ relation. Let $s_0, s_1, \ldots, s_{k'}$, where $k' > k-1$, be a topological sort of $\mathrm{domain}(\Pi)$. The initial state s_0 has no incoming edges, and therefore we can assume w.l.o.g. that it is the first state in the topological sort (if it is not first, we can move it to the front without invalidating the topological sort.)

Construct a satisfying assignment v for $\mathsf{CMP}_{M,A,k}$ as follows: for all $i \in \{0, \ldots, k-1\}$, set

- $v(u_i) = s_i$,
- $v(x_i^q) = 1$ if $s_i \in I_q$, and otherwise $v(x_i^q) = 0$,
- $v(\rho_i^q) = \rho_q(s_i)$,
- For $q \in Q_\Diamond$, set $v(t_i^q) = PS_q(s_i)$ if $s_i \in I_q$, and $v(t_i^q) = s_0$ otherwise. (The choice of s_0 if $s_i \notin I_q$ is arbitrary; any model state will serve.)

To show that $v \models \mathsf{CMP}_{M,A,k}$, consider the constraints that appear in $\mathsf{CMP}_{M,A,k}$.

- Partial proof:

$$\mathsf{PPRF}_{M,A,k} = \mathsf{I}(u_0) \wedge x_0^{q_0} \wedge \bigwedge_{i=0}^{k-1} \bigwedge_{q \in Q} x_i^q \rightarrow \langle\!\langle \delta(q) \rangle\!\rangle_i^{\mathsf{partial}}$$

  As in the proof of Theorem 3.1.1, the satisfaction of the Initiality constraint and the Invariance and Progress constraints, for all $q \in Q \setminus Q_\Diamond$, follows from the fact that Π is a valid proof.

  For $q \in Q_\Diamond$ and i such that $v(x_i^q) = 1$, $s_i \in I_q$, and thus $PS_q(s_i)$ is defined. Suppose $PS_q(s_i) = s_j$. By the construction of v, $v(t_i^q) = s_j$. There are two possibilities:

  1. $j \leq k-1$, in which case $v(u_j) = s_j$, and the Invariance and Progress constraint is satisfied: there is a transition from s_i to s_j and therefore $v \models \mathsf{R}(u_i, t_i^q)$, and also since s_j is a proof successor for s_i and q, $v \models x_j^{q_1}$ and $v \models \mathsf{P}_q(\rho_j^{q_1}, \rho_i^q)$.
  2. $j \geq k$, in which case for all $\ell \in \{0, \ldots, k-1\}$, $v(u_\ell) = s_\ell \neq s_j$. (A state does not appear twice in a topological sort, and therefore all the states that precede s_j are

different from it.) In this case the weakened Invariance and Progress constraint is also satisfied, this time weakly: for all $\ell \in \{0, \ldots, k-1\}$, $v(t_i^q) = s_j \neq s_\ell = v(u_\ell)$, and the implication is satisfied.

- Conciseness:

$$\mathsf{CON}_{A,k} = \bigwedge_{i=1}^{k-1} \left(\bigvee_{j=0}^{i-1} \bigvee_{q \in Q_\Diamond} (u_i = t_j^q) \wedge x_j^q \right)$$

  Let $i \in \{1, \ldots, k-1\}$. We know that s_i is necessary in Π, and hence it is reachable from s_0 by dependency edges of the form $(s, PS_q(s))$. In particular, there exists a shortest parth from s_0 to s_i. Let (s_j, s_i) be the last edge on such a path. From the definition of $\mathrm{REQ}_{\Pi,PS}$, $(s_j, s_i) \in \mathrm{REQ}_{\Pi,PS}$. Hence, $j < i$ (in the topological sort) and this implies that $j < k-1$. Thus, there is a $j < i$ and $q \in Q_\Diamond$ such that $v(u_i) = s_i = PS_q(s_j) = v(t_j^q)$. Also, $v(x_j^q) = 1$, because $s_j \in I_q$ (otherwise $PS_q(s_j)$ would not be defined). The disjunction is therefore satisfied.

- Distinctness:

$$\mathsf{D}_k = \bigwedge_{0 \leq i \leq k-1} \left(\bigwedge_{i < j \leq k-1} u_i \neq u_j \right)$$

  This constraint is satisfied because no state appears twice in a topological sort.

□

In the theorem, we require only one direction in the implication: if there exists a proof with more than k states, then our dynamic completeness criterion must not falsely indicate that it is safe to halt at k and conclude that M does not satisfy A. This is a soundness result. We cannot require the other direction, because if we had a formula that provided both directions of the implication, it would immediately be able to tell us whether or not M satisfies A, even when the bound was 1! Indeed, satisfiability of $\mathsf{CMP}_{M,A,k}$ only indicates that there *may* be a proof with more than k states. Whether or not such a proof actually exists can only be determined at greater bounds. However, the following lemma shows that $\mathsf{CMP}_{M,A,k}$ will become unsatisfiable when there does not exist a non-wasteful partial proof of size k.

Lemma 4.2.4. *If $\mathsf{CMP}_{M,A,k}$ is satisfiable, then there exists a partial proof $\Upsilon = (I, \rho)$ with $|\operatorname{domain}(\Upsilon)| = k$ which is PS-concise w.r.t. some proof-successor assignment PS.*

Proof. Let v be a satisfying assignment for $\mathsf{CMP}_{M,A,k}$. Construct a partial proof Υ as usual, by setting $I_q = \{v(u_i) \mid v(x_i^q) = 1\}$ and $\rho_q(v(u_i)) = v(\rho_i^q)$ for all $q \in Q$ and $i \in \{0, \ldots, k-1\}$.

Define a proof-successor assignment PS for Υ by $PS_q(v(u_i)) = v(t_i^q)$ if $v(u_i) \in I_q$ and $v(t_i^q) = v(u_j)$ for some $j \in \{0, \ldots, k-1\}$; otherwise, $PS_q(v(u_i))$ is undefined.

Υ is a partial proof: the Invariance and Progress constraints for all $q \in Q \backslash Q_\Diamond$ are the same in $\mathsf{CMP}_{M,A,k}$ as they would be in $\mathsf{PRF}_{M,A,k}$, so the proof of Theorem 3.1.1 applies. For $q \in Q_\Diamond$ with $\delta(q) = \Diamond q_1$ and $s \in I_q$, there is some $i \in \{0, \ldots, k-1\}$ such that $s = v(u_i)$ and $v(x_i^q) = 1$. Therefore $v \models \langle\!\langle \Diamond q_1 \rangle\!\rangle_i^{\mathsf{partial}}$, that is, $v \models \mathsf{R}(u_i, t_i^q) \wedge \bigwedge_{j=0}^{k-1} \left((t_i^q = u_j) \rightarrow \left(x_j^{q_1} \wedge \mathsf{P}_q(\rho_j^{q_1}, \rho_i^q)\right)\right)$. If there is no $j \in \{0, \ldots, k-1\}$ such that $v(t_i^q) = v(u_j)$, then $v(t_i^q)$ is a fresh successor for $v(u_i)$ which is not one of the regular proof states. If there is some $j \in \{0, \ldots, k-1\}$ such that $v(t_i^q) = v(u_j)$, then $v(u_j)$ is a proof successor for $v(u_i)$ and q. Either way, the Weak Invariance and Progress requirement for s and q is satisfied.

Υ is also PS-concise: we will show by induction on i that $v(u_i)$ is necessary w.r.t. PS. For $i = 0$, $v(u_i) = s_0$ (from Initiality), and s_0 is always necessary in any proof. Now suppose $v(u_j)$ is necessary for all $j < i$, and let us show that $v(u_i)$ is also necessary.

Among the other constraints in $\mathsf{CMP}_{M,A,k}$, v satisfies the Conciseness constraint:

$$\mathsf{CON}_{A,k} = \bigwedge_{i=1}^{k-1} \left(\bigvee_{j=0}^{i-1} \bigvee_{q \in Q_\Diamond} (u_i = t_j^q) \wedge x_j^q \right)$$

Therefore, by the construction of Υ, there exist some $j < i$ and $q \in Q_\Diamond$ such that $v(u_j) \in I_q$ and $v(t_j^q) = v(u_i)$. Therefore, $PS_q(v(u_j))$ is defined, and its value is $PS_q(v(u_j)) = v(u_i)$. From the induction hypothesis, $v(u_j)$ is necessary, and since we have just shown a dependency from $v(u_j)$ to $v(u_i)$ we obtain that $v(u_i)$ is necessary as well. □

The scheme for using the dynamic completeness criterion is shown in Alg. 4, which takes as input a $\Diamond$-automaton A and a structure M, and returns **true** iff M satisfies A. The correctness of the algorithm follows from Theorem 4.2.1.

Algorithm 4 BMC using the dynamic completeness criterion

```
function bmc(M, φ)
A ← to-automaton(¬φ)
for k = 1 to |S| do
  res ← SAT-solve(PRF_{M,A,k})
  if res = sat then
    return false
  end if
  res ← SAT-solve(CMP_{M,A,k})
  if res = unsat then
    return true
  end if
end for
return true
```

Remark (On symmetry-breaking). In Section 3.2, we discussed several ways to improve the encoding of $\mathsf{PRF}_{M,A,k}$. One of the improvements, encoding successors explicitly (Sec-

tion 3.2.2), was used directly in the construction of $\mathsf{CMP}_{M,A,k}$. The others can also be applied to of $\mathsf{CMP}_{M,A,k}$, with one exception: symmetry-breaking (Section 3.2.1). The proof-successor order $\mathrm{REQ}_{\Pi,PS}$, used in the proof of Theorem 4.2.1, imposes its own order on the proof states; it is no longer true that all the states except s_0 are interchangeable, because they must appear in topological sort according to $\mathrm{REQ}_{\Pi,PS}$. We do get a degree of symmetry-breaking from using the $\mathrm{REQ}_{\Pi,PS}$ order, but it is not as complete as we had in Section 3.2.1, since there may be states that are independent of each other w.r.t. $\mathrm{REQ}_{\Pi,PS}$ and can take any order between themselves.

Remark (Future work). The dynamic completeness criterion we presented can be improved by strengthening the analysis of which states are really "necessary". Currently, we say that a state is "necessary" if there is a chain of proof obligations for $Q_\Diamond$ states that requires it. However, this does not take into account the fact that we are allowed to spuriously add states to the invariants of $Q_\Diamond$ states even when the proof does not require it, thus adding dependencies and making states seem necessary when they might otherwise not be. A better criterion would take into account not only proof obligations for $Q_\Diamond$, but all proof obligations, and it would not allow model states to belong to invariants in which they are not necessary to satisfy some proof obligations. However, it is not obvious how this should be done for disjunction: if $s \in I_q$ where $\delta(q) = q_1 \vee q_2$, is s necessary in I_{q_1}, or in I_{q_2}, or both? Can we require that s be in either I_{q_1} or I_{q_2} but not both, or would doing so affect the soundness of the completeness criterion? This question must be addressed before developing a completeness criterion that takes into account all proof obligations.

We are also interested in developing a static completeness criterion for all weak automata, not only ECTL.

Chapter 5

A Comparison of the Three Approaches to BMC for Branching-Time Logic

In this section we will discuss, somewhat informally, various aspects of the BMC encodings of [33], [42] and our own encoding, and their relative merits in terms of efficiency and the counter-examples they produce. An empirical evaluation of the performance of our approach as compared to [42] is presented in Chapter 6.

5.1 The path-based encoding

For the sake of simplicity, we will discuss the path-based encoding of [33] for ACTL. The encoding is extended to ACTL* in [45], but the extension involves more technical details and does not introduce any new ideas.

In the encoding of [33], the counter-example is represented as a collection of paths of length k. The number of paths used for a formula φ is given by $f_k(\varphi)$, where $f_k(\varphi)$ is defined inductively:

$$
\begin{aligned}
&f_k(p) = f_k(\neg p) = 0 \text{ for all } p \in AP \\
&f_k(\varphi_1 \wedge \varphi_2) = f_k(\varphi_1) + f_k(\varphi_2) \\
&f_k(\varphi_1 \vee \varphi_2) = \max(f_k(\varphi_1), f_k(\varphi_2) \\
&f_k(EX\varphi_1) = 1 + f_k(\varphi_1) \\
&f_k(EG\varphi_1) = (k+1) \cdot f_k(\varphi) + 1 \\
&f_k(E[\varphi_1 \, U \, \varphi_2]) = k \cdot f_k(\varphi_1) + \max(f_k(\varphi_1), f_k(\varphi_2)) + 1
\end{aligned}
$$

The total number of model states one has to encode for a formula φ when the bound is k

is $O(k^d)$, where d is the nesting depth of temporal operators in φ; that is, the encoding is polynominal in the bound k, but it is exponential in the nesting depth of temporal operators in φ (specifically, EG and EU operators).

The encoding itself is defined as follows. Let $u_{i,j}$ denote the jth state on the ith path encoded. For each subformula ψ of φ, we define a Boolean formula $[\![\psi, u]\!]_k$, which informally means "ψ is satisfied by u". The formula is constructed depending on the main operator in ψ:

$$
\begin{aligned}
&[\![p, u]\!]_k = \mathsf{L}_p(u) \\
&[\![\neg p, u]\!]_k = \neg \mathsf{L}_p(u) \\
&[\![\varphi_1 \wedge \varphi_2, u]\!]_k = [\![\varphi_1, u]\!]_k \wedge [\![\varphi_2, u]\!]_k \\
&[\![\varphi_1 \vee \varphi_2, u]\!]_k = [\![\varphi_1, u]\!]_k \vee [\![\varphi_2, u]\!]_k \\
&[\![EX\varphi_1, u]\!]_k = \bigvee_{i=1}^{f_k(\varphi)} ((u = u_{i,0}) \wedge [\![\varphi_1, u_{i,1}]\!]_k) \\
&[\![EG\varphi_1, u]\!]_k = \bigvee_{i=1}^{f_k(\varphi)} \left((u = u_{i,0}) \wedge \bigwedge_{j=0}^{k} [\![\varphi_1, u_{i,j}]\!]_k \wedge \bigvee_{\ell=0}^{k} \mathsf{R}(u_{i,k}, u_{i,\ell}) \right) \\
&[\![E[\varphi_1 \, U \, \varphi_2, u]\!]_k = \bigvee_{i=1}^{f_k(\varphi)} \left((u = u_{i,0}) \wedge \bigvee_{j=0}^{k} \left([\![\varphi_2, u_{i,j}]\!]_k \wedge \bigwedge_{\ell=0}^{j-1} [\![\varphi_1, u_{i,\ell}]\!]_k \right) \right)
\end{aligned}
$$

Simply put, to show that u satisfies ψ, one stipulates, as a disjunction over all the paths, the existence of a path indexed i starting at u ($u = u_{i,0}$) which serves as a witness for ψ. For example, to show that u satisfies $EG\psi$, we require that one of the paths be lasso-shaped and contain only states that satisfy ψ.

In addition to these constraints, the paths are all constrained to be paths of the model:

$$
[\![M]\!]_k = \bigwedge_{i=1}^{f_k(\varphi)} \bigwedge_{j=0}^{k-1} \mathsf{R}(u_{i,j}, u_{i,j+1})
$$

and in addition, the first path must be an initialized path. The resulting encoding is given by

$$
[\![M, \varphi]\!]_k = \mathsf{I}(u_{1,0}) \wedge [\![M]\!]_k \wedge [\![\varphi, u_{1,0}]\!]_k
$$

In Example 5.3.1 we show how the encoding is applied to disprove the formula $AFAG\neg p$.

Counter-examples

It is not possible to tell in advance which path will attach to which point in which other paths. The disjunction in the translation of formulas with temporal operators allows each of the $f_k(\varphi)$ paths to serve as a witness required at some point in some other path, and in

fact one path may serve to satisfy different requirements.

The counter-example obtained by the approach of [33] suffers from several drawbacks. First, it is not necessarily minimal in size in any sense — neither in the number of states encoded nor in the number of paths encoded, since $f_k(\varphi)$ is only an upper bound on the number of paths that may be needed. Second, it may contain parts that are irrelevant to the property, and in fact requiring all the paths to be of the same length makes this outcome very likely. Third, there is no way to tell which parts of the counter-example are relevant to which subformulas in the property, or indeed which parts are relevant at all. For example, if we set out to disprove a conjunction $\varphi_1 \wedge \varphi_2$, the resulting counter-example will not enable us to determine which of the formulas φ_1 and φ_2 was falsified, unless we model-check the counter-example to see whether it satisfies $\neg\varphi_1$ or $\neg\varphi_2$.

One advantage of the path-based encoding over the tree-based encoding which will be discussed next is that it allows "re-use" of counter-example states. If we have a path $u_{i,0}u_{i,1}\ldots u_{i,k}$ that serves as a witness to show that $u_{i,0}$ satisfies ψ, we can "use" this path every time we need to show that $u_{i,0}$ satisfies ψ, which may occur more than once. However, the encoding does not enforce re-use of paths, so it is entirely possible that two different paths, from the same model state, will be used to justify the satisfaction of the same subformula.

5.2 The tree-based encoding

The encoding presented in [42] uses as its basis Stirling's local proof rules from [38]. We will give a brief and informal overview of the proof system.

To make the proof-rules local, the μ-calculus syntax is enriched with memoryfull fixpoint operators $\nu X\,\{r_0,\ldots,r_m\}\,.\psi$ and $\mu X\,\{r_0,\ldots,r_m\}\,.\psi$, called *tagged fixpoint operators*, which "remember" the states already visited in the fixpoint. Formally, $[\![\mu X\,\{r_0,\ldots,r_m\}\,.\psi]\!]^e_M$ is the least fixpoint of the function $\tau : 2^S \to 2^S$ defined by $\tau(Z) = [\![\psi]\!]^{e[X\leftarrow Z]}_M \setminus \{r_0,\ldots,r_m\}$; and $[\![\nu X.\psi]\!]^e_M$ is the greatest fixpoint of the function $\tau : 2^S \to 2^S$ defined by $\tau(Z) = [\![\psi]\!]^{e[X\leftarrow Z]}_M \cup \{r_0,\ldots,r_m\}$.

The intuition for the way tagged fixpoint operators are used in Stirling's proof rules can be seen from the Reduction Lemma ([24], [44]).

Lemma 5.2.1 (Reduction Lemma)**.** *Given a monotonic function $\tau : 2^S \to 2^S$, for all $s \in S$,*

(i) $s \in \mu X.\tau(X)$ *iff* $s \in \tau\,(\mu X.\,(\tau(X) \setminus \{s\}))$*, and*

(ii) $s \in \nu X.\tau(X)$ *iff* $s \in \tau\,(\nu X.\,(\tau(X) \cup \{s\}))$*.*

Expressed in terms of the tagged fixpoint operators, the Reduction Lemma says that $s \in [\![\mu X.\psi]\!]^e_M$ iff $s \in [\![\psi]\!]^{e[X\leftarrow\mu X\{s\}.\psi]}_M$, and $s \in [\![\nu X.\psi]\!]^e_M$ iff $s \in [\![\psi]\!]^{e[X\leftarrow\nu X\{s\}.\psi]}_M$. Therefore, to prove that s satisfies $\mu X.\psi$ it is sufficient to show that s satisfies $\psi[X \leftarrow \mu X\,\{s\}\,.\psi]$ and

similarly for greatest fixpoints. These are two of the proof rules in the proof system of [38], and they are called "μ-unroll" and "ν-unroll", respectively. We refer to each application of one of these rules as an *unrolling*.

Another proof rule for greatest fixpoints in [38] is actually an axiom: from no premises, it is possible to conclude that s satisfies $\nu X \{r_0, \ldots, r_m, s\} . \psi$. In addition to this axiom and the unrolling rules, the proof system contains standard proof rules for all the Boolean connectives; e.g., to prove $\psi_1 \wedge \psi_2$, one must prove ψ_1 and ψ_2.

The encoding in [42] expresses the proof rules as a Boolean formula. The bound limits the *depth* of the proof, which is the number of unrollings the proof uses along each branch. The encoding uses Boolean variables c_j to indicate whether we are looking for a proof, or whether we are applying the dynamic completeness criterion and looking for a *beginning* of a proof. When we are looking for a proof, the c_j variables are constrained to zero. In the dynamic completeness criterion they are constrained to one.

For each subformula ψ of φ, state u and integer d, we define a Boolean formula $[\![\varphi, u, d]\!]$, whose satisfaction will correspond to the fact that u satisfies φ, and it requires at most d applications of the μ- and ν-unroll proof rules to show this. The encoding is given below.

$$[\![\nu X \{v_0, \ldots, v_m\} . \psi, u, d]\!] =$$

$$= \begin{cases} \bigwedge_{i=0}^{m} (u \neq v_i) \leftrightarrow c_j & d = 0 \\ \bigvee_{i=0}^{m} (u = v_i) \vee [\![\psi[X \leftarrow \nu X \{v_0, \ldots, v_m, u\}], u, d-1]\!] & d > 0 \end{cases}$$

where c_j is a fresh Boolean variable.

$$[\![\mu X \{v_0, \ldots, v_m\} . \psi, u, d]\!] =$$

$$= \begin{cases} \bigwedge_{i=0}^{m} (u \neq v_i) \wedge c_j & d = 0 \\ \bigwedge_{i=0}^{m} (u \neq v_i) \wedge [\![\psi[X \leftarrow \mu X \{v_0, \ldots, v_m, u\}], u, d-1]\!] & d > 0 \end{cases}$$

where c_j is a fresh Boolean variable.

$$[\![\varphi_1 \wedge \varphi_2, u, d]\!] = [\![\varphi_1, u, d]\!] \wedge [\![\varphi_2, u, d]\!]$$

$$[\![\varphi_1 \vee \varphi_2, u, d]\!] = [\![\varphi_1, u, d]\!] \vee [\![\varphi_2, u, d]\!]$$

$$[\![\Diamond \psi, u, d]\!] = \mathsf{R}(u, u') \wedge [\![\psi, u', d]\!]$$

$$[\![p, u, d]\!] = \mathsf{L}_p(u)$$

$$[\![\neg p, u, d]\!] = \neg \mathsf{L}_p(u)$$

The encoding as presented in [42] is more complicated than our presentation here, because it is not assumed that the formula is in negation-normal form.

The formula one uses to search for a counter-example is

$$\Theta(M, \varphi, d) = [\![\varphi, u, d]\!] \wedge (u = s_0) \wedge \bigwedge_j \neg c_j$$

and the formula one uses for the dynamic completeness criterion is

$$\Gamma(M, \varphi, d) = [\![\varphi, u, d]\!] \wedge (u = s_0) \wedge \bigwedge_j c_j$$

Intuitively, to justify that a state u satisfies a fixpoint formula φ, the encoding unrolls the fixpoint until the bound is reached. Then, at the end of the branch, the requirement depends on the value of c_j. If it is zero, indicating that we are looking for a proof, then each proof branch must succeed in showing that u satisfies φ. For a least fixpoint formula, which typically corresponds to liveness, this means that all states along the branch should be distinct and the liveness obligation should be discharged before we reach the end of the branch; therefore, if c_j is zero, the end of the branch is always false (c_j appears as a conjunct in the end-branch encoding, $[\![\mu X \{v_0, \ldots, v_m\} . \psi, u, 0]\!]$). If we have reached the end of the branch we have not discharged the obligation. For a greatest fixpoint formula, which typically corresponds to safety or invariance, a value of 0 for c_j means that the states we visited along the branch must close a loop.

If c_j takes the value 1, then we are only looking for a beginning of a proof. In this case we allow branches corresponding to least fixpoints to end without discharging their obligation, as long as all the states along the branch are distinct, and we allow branches corresponding to greatest fixpoints to end without closing a loop. Such "open-ended" branches may be extended into valid branches when the bound on the number of unrollings is increased.

Unsoundness of the encoding from [42]

The encoding as presented above is unsound: the formula created for the completeness criterion is too strong, and it may be unsatisfiable even when there exists a structure that can be extended into a proof, as shown in the following example.

Example 5.2.1. Consider the ECTL property $EG\, EF\, \mathbf{true}$, expressed in μ-calculus as $\varphi = \nu X. (\mu Y. \mathbf{true} \vee \Diamond Y) \wedge \Diamond X$. Let $M = (\{s_0\}, s_0, \{(s_0, s_0)\}, \emptyset)$ be a model containing only one state, with a self-loop.

Clearly, $M \models \varphi$. To show this in a Stirling-style proof we would need two unrollings: one unrolling of the outer fixpoint, and one unrolling of the inner fixpoint. Hence we can expect that when the bound is 1, we will not find a counter-example, and indeed the formula we obtain with $d = 1$ is

$$[\![\varphi, u_0, 1]\!] = \mathbf{false} \vee [\![(\mu Y. \mathbf{true} \vee \Diamond Y) \wedge \Diamond (\nu X \{u_0\} . (\mu Y. \mathbf{true} \vee \Diamond Y) \wedge \Diamond X), u_0, 0]\!] \equiv$$

$$\equiv [\![(\mu Y.\mathbf{true} \vee \Diamond Y), u_0, 0]\!] \wedge [\![\Diamond (\nu X \{u_0\} . (\mu Y.\mathbf{true} \vee \Diamond Y) \wedge \Diamond X), u_0, 0]\!] \equiv$$
$$\equiv \Big(\mathbf{true} \wedge c_0\Big) \wedge \Big(\mathsf{R}(u_0, u_1) \wedge [\![(\nu X \{u_0\} . (\mu Y.\mathbf{true} \vee \Diamond Y) \wedge \Diamond X), u_1, 0]\!]\Big) \equiv$$
$$\equiv c_0 \wedge \mathsf{R}(u_0, u_1) \wedge ((u_0 \neq u_1) \leftrightarrow c_1)$$

As we expected, $\Theta(M, \varphi, 1) = [\![\varphi, u_0, 1]\!] \wedge (u_0 = s_0) \wedge \neg c_0$ is unsatisfiable, because c_0 and $\neg c_0$ both appear as a conjuncts. Since we know a proof *does* exist, we expect that $\Gamma(M, \varphi, 1) = [\![\varphi, u_0, 1]\!] \wedge (u_0 = s_0) \wedge c_0$ will be satisfiable. But $\Gamma(M, \varphi, 1)$ is not satisfiable: since $\Gamma(M, \varphi, 1)$ constrains c_1 to 1, any satisfying assignment must also satisfy $(u_0 \neq u_1)$; this is impossible, because the only possible assignment for u_0 and for u_1 is s_0. The completeness criterion is unsatisfiable, which indicates that no beginning for a proof exists.

Hence, when the encoding from [42] is used to model-check whether $M \stackrel{?}{\models} \neg\varphi$, the model-checker concludes that $M \models \neg\varphi$ when this is in fact not the case.

The source of the problem is the end-branch behavior for greatest fixpoints:

$$[\![\nu X \{v_0, \ldots, v_m\} . \psi, u, 0]\!] = \bigwedge_{i=0}^{m} (u \neq v_i) \leftrightarrow c_j$$

When we constrain c_j to 1, we are *requiring* the proof branch to be "open-ended" and not close a loop. In the general case, as in the example, it may be that the proof branch has to close a loop in any satisfying assignment, because its length exceeds the diameter of the model.

A simple solution is to drop the constraint $\bigwedge_j c_j$ in $\Gamma(M, \varphi, d)$, leaving the dynamic completeness criterion

$$\Gamma(M, \varphi, d) = [\![\varphi, u, d]\!] \wedge (u = s_0)$$

Counter-examples

The counter-examples produced by the tree-based encoding from [42] suffer from the same disadvantages inherent in the path-based encoding. They are uninformative, in the sense that they can contain states that are not relevant to the property, and there is no way to tell which states are relevant and which are not. Further, in the encoding of [42], each subgoal $[\![\varphi, u, d]\!]$ is allocated its own separate proof subtree, with separate model states. Unlike [33], there can be no re-use of model states and information; if we need to justify $[\![\varphi, u, d]\!]$ in two different places, the SAT solver will have to do the work twice.

5.3 An example

To illustrate the differences between the three approaches, consider the following example.

Example 5.3.1. Let $\varphi = EGEFp$. φ is an ECTL property, and it is equivalent to the $\Diamond L_\mu$ property $\varphi' = \nu X.\,((\mu Y.p \vee \Diamond Y) \wedge \Diamond X)$.

Fig. 5.1 shows the shape of the counter-example that would be encoded in the path-based encoding of [33], with $k = 3$. Recall that to justify that a state u satisfies a subformula ψ, the encoding of [33] may require the existence of a path i, with $u_{i,0} = u$, which serves as a "witness" for ψ. For example, to show that $M, u \models EFp$, we require a path i with $u_{i,0} = u$, such that for some $j \geq 0$ we have $M, u_{i,j} \models p$. These requirements are indicated by dashed lines in Fig. 5.1: a dashed line between two states u and v, where v is the first in a path, indicates that $u = v$ and the path from v serves as a witness for the satisfaction of some formula at u. Note that the dashed lines indicate only one possible way to satisfy the requirements: the disjunction in the encoding of [33] allows any path to be chosen, and in Fig. 5.1 we indicate one possible choice.

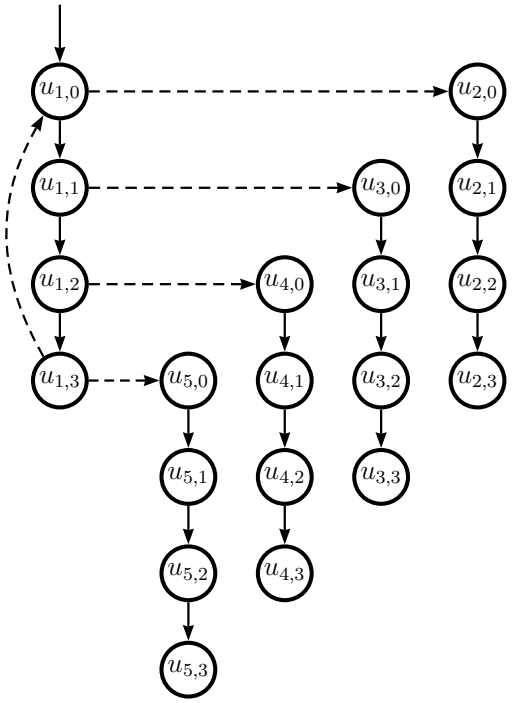

Figure 5.1. The path-based encoding for φ with $k = 3$, using $f_3(\varphi) = 5$ paths

Fig. 5.2 shows the shape encoded in the tree-based encoding of [42] with $k = 3$. The figure shows only the model states and edges, and does not depict the accompanying Boolean constraints. For example, the formula $EFp \equiv \mu Y.p \vee \Diamond Y$ is translated in [42] into a disjunction: either p is satisfied at the current state, or there exists a successor which satisfies EFp. As a result, many of the edges shown in the figure may not exist in the counter-example; e.g., the edge between the states labeled u and v in the figure may not exist if $M, u \models p$. Nevertheless, the encoding includes all the states, and a satisfying assignment does not enable us to determine which edges exist in the model and which do not.

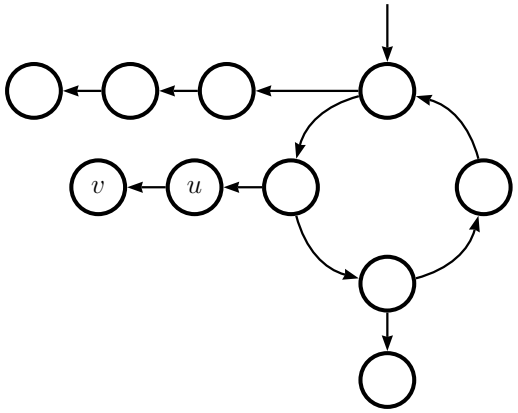

Figure 5.2. The tree-based encoding for φ' with $k = 3$

Consider the model M depicted in Fig. 5.3, which satisfies φ. In our encoding, a counter-example will be found with a bound of 5; it will be M itself. In the path-based encoding of [33], a counter-example will be found when the bound is 4, and it will contain $f_4(\varphi)\cdot(4+1) = 30$ state copies. In the tree-based encoding of [42], the counter-example will be found when the bound is 8, and it will contain 44 state copies. In this example, since the model only contains states that are relevant to the counter-example, all the states encoded in all cases will be copies of the same five model states. However, in a more realistic setting, the counter-example is likely to be much smaller than the model, and states that are not relevant to the counter-example are likely to be returned by the encodings of [33] and [42]. (Our experimental results, presented in Chapter 6, show this very clearly.)

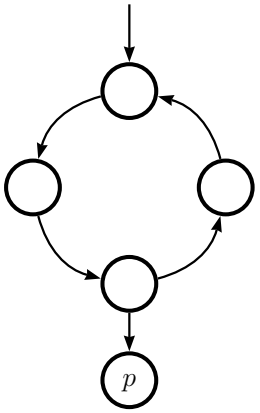

Figure 5.3. The model from Example 5.3.1

5.4 Discussion

The graph-based encodings presented in Chapter 3 do not suffer from the disadvantages of the path-based and tree-based encodings. The counter-example is encoded in a very compact

way, with every state appearing at most once. A single encoded model state can serve to justify different requirements; the bits x_i^q for all $q \in Q$ serve as annotation for the counter-example, and allow the SAT solver to "remember" that it has already shown (or assumed) that the ith model state satisfies q and re-use this information later.

In the encodings of [33] and [42], a counter-example will only be found when the bound reaches the length of the longest part of the counter-example (or proof tree, for [42]). If there exists a counter-example where some of the branches are short but some are long, we will not find it until the bound is large enough to encompass the longest branches. Since all branches encoded have the same length, this causes the path-based and tree-based encodings to encode unnecessary states along some branches. In addition to the wastefulness inherent in these encodings, it is impossible to distinguish between necessary model states and states that were added to the counter-example to pad short branches. The graph-based encoding we presented does not have this drawback: it will find a counter-example as soon as enough states have been encoded, and no unnecessary states are added.

As for complexity, the number of variables used in the graph-based encoding grows as $O(kn \log kn)$, where n is the number of bits needed to encode a model states and k is the bound. This complexity in the bound seems better than the other two encodings, but this is actually not a fair comparison, because the bound does not have the same meaning in all the encodings. In the other two encodings the bound corresponds roughly to the height of the tree encoded, and in our encoding it is exactly the number of states. Thus, when the bound increases by 1, we add only one state to our encoding, but we add a full layer to the path-based and tree-based encodings. As a result, a counter-example may be found when the bound is smaller using the other two encodings, especially if the counter-example is a shallow and dense tree.

As a result of the difference in the meaning of the bound, the static completeness thresholds of the various encodings are incomparable. However, it is worth noting that the graph-based approach does not suffer from one drawback that is common to both linear-time BMC and the path-based and tree-based encodings: in the general case, the diameter of the model, or even the number of states, is not a sound completeness threshold for these encodings [8], as shown in the following example.

Example 5.4.1. Let k be an odd natural number, and consider a model that consists of k states lined up in one k-cycle: $M_{k\text{-cycle}} = (\{0, \ldots, k-1\}, 0, \{(i, (i+1) \mod k) \mid 0 \leq i \leq k-1\}, L)$ over $AP = \{p\}$, where $L(0) = \{p\}$ and $L(s) = \emptyset$ for all $s \neq 0$. The model is shown in Fig. 5.4 for $k = 5$.

Let φ be the $\Diamond L_\mu$ property "p is reachable from s_0 in a finite path of even non-zero length": $\varphi = \Diamond\Diamond\mu X.\,(p \vee \Diamond\Diamond X)$. (In this context we take the length of a path to be the number of edges it contains.) Clearly, $M \models \varphi$, as witnessed by the path $\pi = 0 \to 1 \to \ldots \to (k-1) \to 0 \to 1 \to \ldots \to (k-1) \to 0$, which contains $2k$ edges. This is the shortest

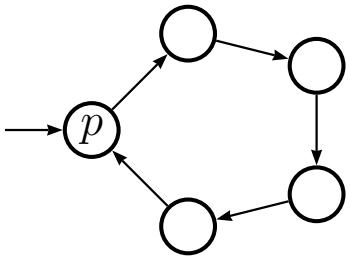

Figure 5.4. The model $M_{k\text{-cycle}}$ from Example 5.4.1

path that can serve as a witness for φ; hence, a BMC method which uses path length (for linear-time) or depth of the tree (for branching-time) as the bound will only find π when the bound reaches $2k$, exceeding both the diameter of the model and the number of states. The tree-based encoding from [42] is an exception: it uses number of unrollings as the bound, and in our example, the number of unrollings necessary to show that $M \models \varphi$ is k. Nevertheless, the counter-example returned in the tree-based encoding will be a path of length $2k$, so the size of the model is still exceeded. (It is not difficult to construct examples in which the bound in the tree-based encoding also exceeds the diameter and number of states in the model.) In contrast, the graph-based encoding will find a witness when the bound reaches k.

Chapter 6

Experimental Results

We implemented a prototype version of our BMC encoding and the encoding of [42] in the NuSMV2 verification framework [7]. Our prototype implementation handles only ACTL properties (without fairness). We compared the encodings on random models and properties.

6.1 The Setup

Following the methodology in [19], we tested the encodings on randomly generated models and random ACTL properties. We used a model-generation algorithm similar to the one used in [19]. Each generated model has only one variable, an integer x ranging over the values 1..100; thus, the models have 100 states each. We use $x = s$ to denote a single state of the model for some $s \in \{1, \ldots, 100\}$.

In generating the models we used a procedure $\mathsf{toss}(p)$ which tosses a coin with probability p of returning 1. The transitions from each state $x = s$ were generated as follows:

1. Randomly choose an integer $t \in \{1, \ldots, 100\}$ and add a transition from $x = s$ to $x = t$ (to ensure at least one transition).

2. While $\mathsf{toss}(p) = 1$, choose another integer $t' \in \{1, \ldots, 100\}$ at random and add a transition from $x = s$ to $x = t'$.

In our experiments we used values of $p = 0.5$ and $p = 0.8$, resulting in models of different diameter: using $p = 0.5$ results in sparse models with large diameter, and using $p = 0.8$ results in dense models with smaller diameter.

The models thus generated are characterized by a very complex transition relation, expressed in SMV as a large switch statement over x.

To test the hypothesis that our encoding is more suitable for complex properties, we tested random properties of varying nesting depths: we consider an ACTL formula to be of nesting depth d if it contains at most d nestings of AU, AW, AF and AG operators. Nested AX

operators and Boolean operators (conjunction and disjunction) do not count in the nesting depth, because they do not contribute to the exponent in the encoding of [42]. (That is, each Boolean operator or AX operator in the formula only increases the size of the tree-based encoding linearly.) The atomic propositions we used were of the form $\bigvee_{i=0}^{r}(x = s_i)$, where r and $s_0, \ldots, s_r$ are randomly generated integers.

We divided the formulas into liveness formulas, safety formulas, and formulas that are neither safety nor liveness formulas (which we refer to as "mixed" formulas). The class of the formula affects the encoding: to disprove a liveness formula — that is, prove a safety formula — our encoding does not need to use ranks. In contrast, to disprove a safety or mixed formula the encoding must use ranks, but the counter-example must close loops in the model, unlike a counter-example to a safety formula.

6.2 The Results

Due to the complex encoding of the transition relation, neither encoding was able to reach large bounds, though in general the encoding from [42] was able to reach larger bounds than ours (and also needed larger bounds to disprove the same properties). We used a maximal bound of 20. Our results are presented in Table 6.2, Table 6.3 and Table 6.1, which show success rates in disproving mixed, liveness and safety properties (respectively). Each table includes the following columns.

- Edge prob.: the probability p of adding another transition to the current transition.
- ND: the nesting depth of the temporal formulas.
- TB: the tree-based encoding from [42], with the completeness criterion disabled.
- GB: the specialized graph-based encoding for ACTL presented in Section 3.3, with the completeness criterion disabled.
- GB+E: the specialized graph-based encoding with successors encoded explicitly (Section 3.2.2), with the completeness criterion disabled.

All variants of our encoding included symmetry-breaking (Section 3.2.1).

6.3 Analysis

The main conclusion we draw from the randomized experiments is that, as expected, our encoding is much less sensitive to the nesting depth of the formula than the tree-based encoding from [42]. The performance of the tree-based encoding declines sharply as the nesting depth increases, whereas the graph-based encoding performs not much worse for

Edge prob.	ND	TB	GB	GB+E
0.2	2	**99%**	81%	83%
	3	**86%**	75%	74%
	4	67%	**72%**	70%
	5	58%	**66%**	59%
0.5	2	**100%**	99%	99%
	3	**97%**	95%	94%
	4	92%	**97%**	94%
	5	71%	**97%**	88%
0.8	2	**100%**	**100%**	**100%**
	3	**100%**	**100%**	**100%**
	4	99%	**100%**	**100%**
	5	93%	**100%**	**100%**

Table 6.1. Success rates in disproving mixed safety/liveness properties

Edge prob.	ND	TB	GB	GB+E
0.2	2	**99%**	72%	75%
	3	**81%**	71%	72%
	4	58%	**68%**	60%
	5	44%	**67%**	51%
0.5	2	**99%**	96%	**99%**
	3	**98%**	97%	96%
	4	81%	**96%**	86%
	5	82%	**99%**	89%
0.8	2	**100%**	**100%**	**100%**
	3	**100%**	**100%**	**100%**
	4	95%	**100%**	**100%**
	5	90%	**100%**	99%

Table 6.2. Success rates in disproving liveness properties

larger nesting depths than it does for small nesting depths. This sensitivity can be explained in several ways:

1. The complexity of the formula encoded in the tree-based encoding is $O(k^d)$, where d is the nesting depth of the formula and k is the bound, whereas in our encoding, the complexity is $O(kd \log kd)$.

2. ACTL formulas with small nesting depth are more "linear-time-like", and it stands to reason that the encoding from [42], which is more similar to the linear-time BMC encodings, does better in these cases. When the formula has a large nesting depth, its branching structure becomes more prominent.

Our encoding appears to result in Booelean formulas that are small and use a small number of variables, but are nevertheless difficult for the SAT solver. We conjecture that the

Edge prob.	ND	TB	GB	GB+E
0.2	2	**100%**	93%	92%
	3	**95%**	88%	84%
	4	**88%**	**88%**	81%
	5	76%	**79%**	71%
0.5	2	**100%**	98%	98%
	3	**100%**	**100%**	98%
	4	**98%**	97%	95%
	5	87%	**99%**	90%
0.8	2	**100%**	**100%**	**100%**
	3	**100%**	**100%**	**100%**
	4	97%	**100%**	**100%**
	5	99%	**100%**	**100%**

Table 6.3. Success rates in disproving safety properties

very compactness of the formulas results in most of the work being left to the SAT solver. The tree-based encoding, which lays out the structure of the counter-example in advance, leads to easier SAT formulas for simple formulas; however, for more complex formulas (nesting depth 4 and 5), the penalty from the large size of the formula appears to exceed the benefit.

We also observed a memory/time tradeoff between the two encodings. The formulas generated in the tree-based encoding are generally very large; for larger nesting depths, the SAT solver sometimes ran out of memory. However, when memory did not run out, the formulas were usually quickly solved. In contrast, the formulas generated in our graph-based encoding are "hard" for the SAT solver, but in all our experiments, the SAT solver never once ran out of memory while solving a formula resulting from our encoding.

Explicit-successors vs. straightforward encoding

Contrary to our expectations, the explicit-successor encoding presented in Section 3.2.2 did not improve performance, and instead seemed to worsen it significantly. We conjecture that adding an explicitly-encoded successor and removing the transition relation from the disjunction hindered the SAT solver's conflict-learning mechanism, by making the relationship between variables less immediate. It is interesting to note, however, that the explicit-successor encoding was sometimes able to solve instances that the regular encoding was not able to solve.

Safety vs. liveness

Again confounding our expectations, the classification of the formulas did not appear to affect the performance of our encoding. The ranks required for disproving safety formulas apparently balance the need to close loops in the model when disproving liveness formulas.

In contrast, it is interesting to note that the tree-based encoding appeared to take a performance hit when disproving mixed formulas.

6.3.1 Counter-examples

In addition to the success rate and time it took to disprove each formula, we also recorded the size of the counter-example returned in each encoding. The counter-examples returned by the tree-based encoding were on average 40 times larger than the counter-examples returned by the graph-based encoding. This supports our hypothesis that due to the worst-case analysis performed in the tree-based encoding of [42], the counter-example returned will contain many unnecessary states, and thus will be much less informative than the counter-examples returned by our graph-based encoding.

Chapter 7

Conclusions

In this work, we presented a novel approach to bounded model-checking for branching-time logic. Our approach is based on translating Namjoshi-style proof obligations to Boolean constraints; it leads to small, informative counter-examples, and improved performance for complex properties. We presented a general encoding for universal alternating parity tree automata, which correspond to universal μ-calculus properties. We then showed how the encoding can be specialized for weak alternating Büchi tree automata, which correspond to the alternation-free fragment of universal μ-calculus.

The encoding allows one to find a counter-example of some bounded size, but not to verify properties. We presented a static completeness threshold for ACTL, and a dynamic completeness threshold for general universal μ-calculus properties, which can both be used to verify properties, by determining when it is safe to stop increasing the bound and conclude that the formula is satisfied.

Our work is closely related to many disciplines in formal verification. The specialized encoding we presented in Section 3.3 can be seen as a Boolean representation of the run of the Emerson-Lei symbolic model-checker for μ-calculus [39]. We use alternating parity tree automata as the specification mechanism, and base our translation to SAT on Namjoshi's proof system for μ-calculus [32]. We also used ideas from the world of linear-time BMC, e.g., [19].

7.1 Future Work

There are several areas where the work presented in this dissertation can be extended and improved. In this section we discuss directions for future research.

7.1.1 Implementation issues

To evaluate our encodings, we created a prototype implementation in the NuSMV2 framework. Our implementation of the encodings from Chapter 3 is naive, and it can probably be improved in the following aspects.

Using a Sequential Circuit SAT solver

In our experiments we used the zChaff SAT solver, an extension of the Chaff solver ([30]). zChaff uses the DPLL algorithm [12], and takes as input a Boolean formula in CNF form. The translation to CNF loses the structure of the original formula, e.g., in our case, the implications between the x_i^q bits and the Invariance and Progress constraints. Recently, a new type of SAT solver has started to gain popularity: Sequential Circuit SAT ([20], [29]). A Sequential Circuit SAT solver takes as input a circuit, and determines whether there exists a sequence of assignments to the circuit inputs under which the circuit's output is 1. If such an assignment exists, the Sequential Circuit SAT solver returns it. The advantage to this approach is that it preserves the circuit's structure and can use it to make informed choices about the order in which it attepts to assign the variables. We believe that a Sequential Circuit SAT solver may be better able to handle the formulas generated in our encodings than traditional DPLL-based solvers.

Ranks and progress

As shown in Chapter 6, the binary encoding of ranks that we used in our encoding is difficult for the zChaff solver to handle. There has been work on using SAT solvers to solve partial order constraints in the context of proving termination (e.g., [11], [3]), and some promising encodings of partial order constraints have emerged. Using these encodings instead of the binary encoding we used may improve performance.

Another option is to use a Satisfiability Module Theories (SMT) solver, for example Yices or CVC3 [40]. An SMT solver can solve Boolean combinations of formulas in some first-order theory; in our case we could use constraints in the theory of partial orders over integers to handle the ranks and the progress requirements.

7.1.2 Improved encodings and applications

The encoding we presented represents one extreme — it is very compact and conveys a lot of information about the counter-example, but it is difficult for SAT solvers to handle; the encoding of [42] represents another extreme, with a lot of "waste" in the encoding of the counter-example and no annotation, but easier SAT instances. It would be interesting to attempt to combine the strengths of the two approaches, for example by adding annotation to the counter-examples generated in [42], or by allowing re-use of proof branches.

We are also interested in pursuing an encoding based on an observation from [27]: to check if a model M satisfies an alternating tree automaton A, one can construct the cross-product $M \times A$ and check its emptiness. $M \times A$ is an alternating automaton on *infinite words*; thus, by employing this approach we go back to the much-explored realm of linear-time BMC. It seems that the encoding of [42] can be interpreted in this light. By casting the problem as a linear-time BMC problem, we avoid the question of how the counter-example should be represented entirely, and also benefit from extensive improvements and optimizations in linear-time BMC since its presentation in 1999 in [4].

It seems our encoding could have applications in bounded synthesis. For example, in [34], bounded synthesis for linear-time specifications is solved through searching for an *annotated* structure of bounded size using a SAT solver. The annotation used in [34] is very similar to Namjoshi-style proof obligations. We believe we can use a similar approach, based on the work presented here, to achieve bounded synthesis for branching-time specifications.

Finally, we believe the ideas we presented in this dissertation may also have an application to linear-time BMC. As explained in Chapter 5, in our encodings it is never necessary to unwind the model more than once, in contrast to most linear-time BMC encodings. It is possible to use our approach to encode a linear-time counter-example as a *simple path*, instead of unwinding a path that may contain multiple loops. It remains to be seen whether this will lead to performance benefits.

Bibliography

[1] Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Solving difficult sat instances in the presence of symmetry. In *DAC '02: Proceedings of the 39th conference on Design automation*, pages 731–736, New York, NY, USA, 2002. ACM.

[2] Mohammad Awedh and Fabio Somenzi. Automatic invariant strengthening to prove properties in bounded model checking. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 1073–1076, New York, NY, USA, 2006. ACM.

[3] Amir M. Ben-Amram and Michael Codish. A sat-based approach to size change termination with global ranking functions. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 218–232. Springer, 2008.

[4] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *TACAS '99: Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 193–207, London, UK, 1999. Springer-Verlag.

[5] Patrick Blackburn, Johan F. A. K. van Benthem, and Frank Wolter. *Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning)*, chapter Automata-theoretic Techniques for Temporal Reasoning. Elsevier Science Inc., New York, NY, USA, 2006.

[6] Marsha Chechik and Arie Gurfinkel. A framework for counterexample generation and exploration. *Int. J. Softw. Tools Technol. Transf.*, 9(5):429–445, 2007.

[7] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *CAV'02*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.

[8] Edmund Clarke, Daniel Kroening, Ofer Strichman, and Joel Ouaknine. Completeness and complexity of bounded model checking. In *VMCAI*, volume 2937 of *LNCS*, pages 85–96, 2004.

[9] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[10] Edmund M. Clarke, Somesh Jha, Yuan Lu, and Helmut Veith. Tree-like counterexamples in model checking. In *LICS'02*, pages 19–29, Washington, DC, USA, 2002. IEEE Computer Society.

[11] Michael Codish, Vitaly Lagoon, and Peter J. Stuckey. Solving partial order constraints for LPO termination. In *RTA*, volume 4098 of *LNCS*, pages 4–18. Springer, 2006.

[12] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[13] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings of the 32nd annual symposium on Foundations of computer science*, pages 368–377, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[14] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FoCS'91*, pages 368–377, 1991.

[15] E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model-checking for fragments of mu-calculus. In *CAV*, volume 697 of *LNCS*, pages 385–396. Springer, 1993.

[16] E. Allen Emerson and Chin-Laung Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract).

[17] Alan M. Frisch, Daniel Sheridan, and Toby Walsh. A fixpoint based encoding for bounded model checking. In *FMCAD '02: Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design*, pages 238–255, London, UK, 2002. Springer-Verlag.

[18] Keijo Heljanko, Tommi Junttila, and Timo Latvala. Incremental and complete bounded model checking for full PLTL. In Kousha Etessami and Sriram K. Rajamani, editors, *Proceedings of the 17th International Conference on Computer Aided Verification (CAV'2005)*, volume 3576 of *LNCS*, pages 98–111, Edinburgh, Scotland, United Kingdom, July 2005. Springer.

[19] Keijo Heljanko, Tommi A. Junttila, Misa Keinänen, Martin Lange, and Timo Latvala. Bounded model checking for weak alternating büchi automata. In *CAV*, volume 4144 of *LNCS*, pages 95–108. Springer, 2006.

[20] M. K. Iyer, G. Parthasarathy, and K.-T. Cheng. Satori - a fast sequential sat engine for circuits. In *ICCAD '03: Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, page 320, Washington, DC, USA, 2003. IEEE Computer Society.

[21] Paul B. Jackson and Daniel Sheridan. A compact linear translation for bounded model checking. *Electron. Notes Theor. Comput. Sci.*, 174(3):17–30, 2007.

[22] David Janin and I. Walukiewicz. Automata for the mu-calculus and related results. In *MFCS (Mathematical Foundations of Comuter Science)*, volume 969 of *LNCS*, pages 552–562. Springer, 1995.

[23] M. Jehle, J. Johannsen, M. Lange, and N. Rachinsky. Bounded model checking for all regular properties. In A. Biere and O. Strichman, editors, *Proc. 3rd Int. Workshop on Bounded Model Checking, BMC'05*, volume 144 of *Electr. Notes in Theor. Comp. Sc.*, pages 3–18. Elsevier, 2005.

[24] Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

[25] Daniel Kroening and Ofer Strichman. Efficient computation of recurrence diameters. In *VMCAI*, pages 298–309, 2003.

[26] O. Kupferman and M. Y. Vardi. Freedom, weakness, and determinism: From linear-time to branching-time. In *LICS '98: Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*, page 81, Washington, DC, USA, 1998. IEEE Computer Society.

[27] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360, 2000.

[28] Timo Latvala, Armin Biere, Keijo Heljanko, and Tommi A. Junttila. Simple bounded ltl model checking. In Alan J. Hu and Andrew K. Martin, editors, *FMCAD*, volume 3312 of *Lecture Notes in Computer Science*, pages 186–200. Springer, 2004.

[29] Feng Lu, Madhu Iyer, ganapathy parthasarathy, and C.-K. Cheng. An efficient sequential sat solver with improved search strategies. In *Design, Automation and Test in Europe (DATE)*, March 2005.

[30] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *DAC*, pages 530–535. ACM, 2001.

[31] A.W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Computation Theory*, volume 208, pages 157–168, 1984.

[32] Kedar S. Namjoshi. Certifying model checkers. In *CAV'01*, volume 2102 of *LNCS*, pages 2–13, Paris, France, July 2001.

[33] Wojciech Penczek, Bozena Wozna, and Andrzej Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundam. Inf.*, 51(1):135–156, 2002.

[34] Sven Schewe and Bernd Finkbeiner. Bounded synthesis. In *Proc. ATVA*, pages 474–488. Springer-Verlag, 2007.

[35] Mary Sheeran, Satnam Singh, and Gunnar Stalmarck. Checking safety properties using induction and a sat-solver. In *FMCAD '00: Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design*, pages 108–125, London, UK, 2000. Springer-Verlag.

[36] Sharon Shoham and Orna Grumberg. A game-based framework for ctl counterexamples and 3-valued abstraction-refinement. *ACM Trans. Comput. Logic*, 9(1):1, 2007.

[37] Maria Sorea. Bounded model checking for timed automata. *Electr. Notes Theor. Comput. Sci.*, 68(5), 2002.

[38] Colin Stirling and David Walker. Local model checking in the modal mu-calculus. *Theor. Comput. Sci.*, 89(1):161–177, 1991.

[39] Robert S. Streett and E. Allen Emerson. The propositional mu-calculus is elementary. In *ICALP'84*, pages 465–472, London, UK, 1984. Springer-Verlag.

[40] Aaron Stump, Clark W. Barrett, and David L. Dill. CVC: A cooperating validity checker. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proceedings of the* 14^{th} *International Conference on Computer Aided Verification (CAV '02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 500–504. Springer-Verlag, July 2002. Copenhagen, Denmark.

[41] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Proceedings of the VIII Banff Higher order workshop conference on Logics for concurrency : structure versus automata*, pages 238–266, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.

[42] Bow-Yaw Wang. Proving forall-mu-calculus properties with SAT-based model checking. In *FORTE'05*, volume 3731 of *LNCS*, pages 113–127. Springer, 2005.

[43] Thomas Wilke. Alternating tree automata, parity games, and modal μ-calculus. *Bull. Soc. Math. Belg.*, 8(2), 2001.

[44] Glynn Winskel. A note on model checking the modal nu-calculus. *Theor. Comput. Sci.*, 83(1):157–167, 1991.

[45] B. Woźna. Bounded Model Checking for the universal fragment of CTL*. *Fundamenta Informaticae*, 63(1):65–87, 2004.

[46] Wenhui Zhang. Verification of actl properties by bounded model checking. In Roberto Moreno-Díaz, Franz Pichler, and Alexis Quesada-Arencibia, editors, *EUROCAST*, volume 4739 of *Lecture Notes in Computer Science*, pages 556–563. Springer, 2007.