A Game-Based Framework for CTL Counterexamples and Abstraction-Refinement

Sharon Shoham

A Game-Based Framework for CTL Counterexamples and Abstraction-Refinement

Research Thesis

Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science in Computer Science

Sharon Shoham

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

KISLEV, 5764 Haifa November, 2003

This Research Thesis was done under the supervision of Orna Grumberg in the Department of Computer Science

I wish to express my deep gratitude to Orna Grumberg for being the best supervisor I could have ever hoped for and so much more than that. I thank Orna for introducing me to the world of academic research and showing me that it can be a lot of fun. More importantly, I thank Orna for her friendship, which made my studies so enjoyable. As this work proves, serious issues are sometimes nothing more than just a game.

The generous financial help of the Technion is gratefully acknowledged

Contents

List of Figures

Abstract

Model checking is an efficient procedure that checks whether or not a given system model fulfills a desired property, described as a temporal logic formula. Yet, as real models tend to be very big, model checking encounters the *state-explosion problem*. One solution to this problem is the use of abstraction, that hides some of the details of the original (concrete) model. In this work we consider the branching time logic CTL (Computation Tree Logic). Our work exploits and extends the game-based framework of CTL model checking for incremental abstraction-refinement and counterexamples. We define a game-based CTL model checking for abstract models over the 3-valued semantics, which can be used for verification as well as refutation. The model checking process of an abstract model may end with an indefinite result, in which case we suggest a new notion of refinement, which eliminates indefinite results of the model checking. This provides an iterative abstraction-refinement framework. This framework is enhanced by an *incremental* algorithm, where refinement is applied only where indefinite results exist and definite results from prior iterations are used within the model checking algorithm. We also define the notion of *annotated counterexamples*, which are sufficient and minimal counterexamples for full CTL. We present an algorithm that uses the game board of the model checking game to derive an *annotated counterexample* in case the examined system model refutes the checked formula.

Notation and Abbreviations

CTL — Computation Tree Logic
SCC — Strongly Connected Component
MSCC — Maximal Strongly Connected Component
KMTS — Kripke Modal Transition System
AP — Set of atomic propositions
Lit — Set of literals (atomic propositions and their negations)
φ — CTL formula
M — Model of a system (a Kripke structure or a KMTS)
M_C — Concrete model (Kripke structure)
M_A — Abstract model (KMTS)
$M \models \varphi$ — The model M satisfies φ
$M \not\models \varphi$ — The model M refutes φ
$[(M_C, s) \models \varphi]$ — Concrete semantics of CTL w.r.t Kripke structures
$[(M_A, s) \overset{\mathbf{2}}{\models} \varphi]$ — 2-valued semantics of CTL w.r.t KMTSs
$[(M_A, s) \overset{\mathbf{3}}{\models} \varphi]$ — 3-valued semantics of CTL w.r.t KMTSs
$\mathbf{tt}$ — true
$\mathbf{ff}$ — false
$\perp$ — indefinite
H — Mixed simulation relation
α — Abstraction function
γ — Concretization function
$G_{M \times \varphi}$ — Game-Graph for M and φ
N — Nodes of the game-graph
E — Edges of the game-graph
χ — Coloring function
$\overline{\chi}$ — Partial coloring function
χ_I — Initial coloring function
$C_{M \times \varphi}$ — Annotated counterexample for M and φ

Chapter 1

Introduction

This work exploits and extends the game-based framework [47] of CTL model checking for counterexample and incremental abstraction-refinement.

The first goal of this work is to suggest a game-based new model checking algorithm for the branching-time temporal logic CTL [11] in the context of abstraction. Model checking is a successful approach for verifying whether a system model M satisfies a specification φ, written as a temporal logic formula. Yet, concrete (regular) models of realistic systems tend to be very large, resulting in the *state explosion problem*. This raises the need for abstraction. Abstraction hides some of the system details, thus resulting in smaller models. Abstractions are usually designed to be *conservative* w.r.t. some logic of interest. That is, if the abstract model satisfies a formula in that logic then the concrete model satisfies it as well. However, if the abstract model does not satisfy the formula then nothing is known about the concrete model.

Two types of semantics are available for interpreting CTL formulae over abstract models. The *2-valued* semantics defines a formula φ to be either true or false in an abstract model. True is guaranteed to hold for the concrete model as well, whereas false may be spurious. The *3-valued* semantics [22] introduces a new truth value: the value of a formula on an abstract model may be *indefinite*, which gives no information on its value on the concrete model. On the other hand, both satisfaction and falsification w.r.t the 3-valued semantics hold for the concrete model as well. That is, while abstractions over 2-valued semantics are conservative w.r.t. only positive answers, abstractions over 3-valued semantics are conservative w.r.t. both positive and negative results. Abstractions over 3-valued semantics thus give precise results more often both for verification and falsification.

Following the above observation, we define a game-based model checking algorithm for abstract models w.r.t. the 3-valued semantics, where the abstract model can be used for both verification and falsification. However, a third case is now possible: model checking may end with an indefinite answer. This is an indication that our abstraction cannot determine the value of the checked property in the concrete model and therefore needs to be refined. The traditional abstraction-refinement

framework [30, 9] is designed for 2-valued abstractions, where false may be a false-alarm, thus refinement is aimed at eliminating false results. As such, it is usually based on a counterexample analysis. Unlike this approach, the goal of our refinement is to eliminate indefinite results and turn them into either definite true or definite false.

An advantage of this work lies in the fact that the refinement is then applied only to the indefinite part of the model. Thus, the refined abstract model does not grow unnecessarily. In addition, model checking of the refined model uses definite results from previous runs, resulting in an *incremental* model checking. Our abstraction-refinement process is complete in the sense that for a finite concrete model it will always terminate with a definite "yes" or "no" answer.

The next goal of our work is to use the game-based framework in order to provide counterexamples for the full branching-time temporal logic CTL. When model checking a model M with respect to a property φ, if M does not satisfy φ then the model checker tries to return a counterexample. Typically, a counterexample is a part of the model that demonstrates the reason for the refutation of φ on M. Providing counterexamples is an important feature of model checking which helps tremendously in the debugging of the verified system.

Most existing model checking tools return as a counterexample either a finite path (for refuting formulae of the form AGp) or a finite path followed by a cycle (for refuting formulae of the form AFp[1]) [10, 11]. Recently, this approach has been extended to provide counterexamples for all formulae of the universal branching-time temporal logic ACTL [13]. In this case the part of the model given as the counterexample has the form of a tree. Other works also extract information from model checking [43, 18, 39, 48]. However, this information is presented in the form of a temporal proof, rather than a part of the model.

In this work we provide counterexamples for full CTL. As for ACTL, counterexamples are part of the model. However, when CTL is considered, we face existential properties as well. To prove refutation of an existential formula $E\psi$, one needs to show an initial state from which *all* paths do not satisfy ψ. Thus, the structure of the counterexample becomes more complex.

Having such a complex counterexample, it might not be easy for the user to analyze it by looking at the subgraph of M alone. We therefore *annotate* each state on the counterexample with a subformula of φ that is false in that state. The annotating subformulae being false in the respective states, provide the reason for φ to be false in the initial state. Thus, the annotated counterexample gives a convenient tool for debugging. We propose an algorithm that constructs an annotated counterexample and prove that it is sufficient and minimal. We also discuss several ways to use and present this information in practice.

Games for CTL model checking [47] is a most suitable framework for our goals. The model checking game is played by two players, ∀belard, the refuter who wants

[1] AGp means "for every path, in every state on the path, p holds", whereas AFp means "along every path there is a state which satisfies p".

to show that $M \not\models \varphi$, and $\exists$loise, the prover who wants to show that $M \models \varphi$. The board of the game consists of pairs (s, ψ) of a model state and a subformula, with the meaning that the satisfaction of ψ in the state s is examined. $\forall$belard proceeds from such a node (s, ψ) to a node that helps refuting ψ on s. $\exists$loise chooses her moves with the intention to prove that s satisfies ψ. All possible plays of a game are captured in the *game-graph*, whose nodes are the elements of the game board and whose edges are the possible moves of the players. The initial nodes are pairs (s_0, φ) where s_0 is an initial state of M. It can be shown that $\forall$belard has a winning strategy (i.e., he can win the game regardless of $\exists$loise moves) iff $M \not\models \varphi$. $\exists$loise has a winning strategy iff $M \models \varphi$.

Model checking is then done by applying a *coloring algorithm* on the game-graph [5]. It colors a node (s, ψ) by T iff $\exists$loise has a winning strategy, which means ψ is true in s. It colors it by F iff $\forall$belard has a winning strategy, which means ψ is false in s. At termination, if all initial nodes are colored T then $M \models \varphi$. If at least one initial node is colored F then $M \not\models \varphi$ and we would like to supply a counterexample.

In our work we add abstraction to the discussion. Concrete models for CTL are state-transition graphs (Kripke structures) in which nodes correspond to states of the system and transitions describe possible moves between states. Abstract models consist of abstract states, representing (not necessarily disjoint) sets of concrete states. In order to be conservative w.r.t. CTL, two types of transitions are required: *may*-transitions which represent possible transitions in the concrete model, and *must*-transitions [33, 16] which represent definite transitions in the concrete model. May and must transitions correspond to over and under approximations, and are needed in order to preserve formulae of the form $AX\psi$ and $EX\psi$, respectively.

We consider the 3-valued semantics of CTL formulae. We would like to maintain the property of the 3-valued semantics that both the positive and the negative answers are definite in the sense that they hold for the concrete model as well. To do so, we allow each player to have two roles in the new 3-valued model checking game. The goal of $\forall$belard is either to refute φ on M or to prevent $\exists$loise from verifying. Similarly, the goal of $\exists$loise is either to verify or to prevent $\forall$belard from refuting. As before, $\forall$belard has a winning strategy iff $M \not\models \varphi$, and $\exists$loise has a winning strategy iff $M \models \varphi$. However, it is also possible that none of them has a winning strategy, in which case the value of φ in M is indefinite.

In order to check φ on the abstract model M, we propose a coloring algorithm over three colors: T, F, and ?. If all the initial nodes of the game-graph are colored by T, then we conclude that $M \models \varphi$. If some initial node of the game-graph is colored by F, we know that $M \not\models \varphi$. Both these results apply to the concrete model as well. Yet, if none of the above holds, meaning that none of the initial nodes is colored by F and at least one of them is colored by ?, we have no definite answer. It is then desirable to refine the abstract model.

We choose a criterion for refinement by examining the part of the game-graph which is colored by ?. Once a criterion for refinement is chosen, the refinement is traditionally done by splitting abstract states throughout the entire abstract model.

That is, while the decision on the criterion for refinement is local, the refinement is global. However, the structure of the game-graph allows us to apply it only to the indefinite part of the model. It also allows us to use definite results that were obtained previously. Thus, previous runs are not wasted and the abstract model does not grow where it is not needed.

Other researchers [22] have suggested to evaluate a property w.r.t the 3-valued semantics by reducing the problem to two 2-valued model checking problems: one for satisfaction and one for refutation. Such a reduction will result in the same answer as our algorithm. Yet, it is then not clear how to guide the refinement, in case it is needed, since at least part of the information about the indefinite portion of the game-graph is lost. Thus, the application to refinement demonstrates the advantage of designing a 3-valued model checking algorithm.

As for our second goal, we propose an algorithm that constructs an *annotated counterexample* in case model checking ends with a negative answer, meaning that the checked property φ is refuted by the examined model M. We first deal with the simpler case where model checking is applied to a concrete model. The construction uses the colored game-graph and starts from an initial node which is colored by F. If the formula in a node n is either $AX\psi$ or $\psi_1 \wedge \psi_2$ then we include in the counterexample one successor of n, which is colored by F. This successor needs to be chosen wisely. If the formula in n is either $EX\psi$ or $\psi_1 \vee \psi_2$ then we include in the counterexample all the successors of n (which are all colored by F). The resulting counterexample is an annotated sub-model of M, with possibly some unwinding, that gives the full reason for the refutation of φ on M.

Having defined the notion of an annotated counterexample, we then discuss the construction of annotated counterexamples when abstract models are used. In the 3-valued case, concretization of an abstract annotated counterexample will never fail since the 3-valued abstraction is conservative w.r.t. negative results as well. Thus, we can use an extension of the concrete algorithm to provide an abstract counterexample and derive from it a concrete one.

To conclude, the main contributions of this work are:

- A game-based CTL model checking for abstract models over the 3-valued semantics, which can be used for verification as well as refutation.
- A new notion of refinement, that eliminates indefinite results of the model checking.
- An incremental model checking within the framework of abstraction-refinement.
- A sufficient and minimal counterexample for full CTL.

1.1 Related Work

Games and Automata

Our work uses a characterization of the CTL model checking problem in terms of two-players games. The game-based approach to model checking was introduced by Stirling [46] as a way of combining the algorithmic approach to model checking and the proof system approach. [47, 32, 31] present model checking algorithms based on games for various temporal logics, including CTL and the alternation-free μ-calculus. The model checking problem is described as a game between a refuter, $\forall$belard, and a prover, $\exists$loise, where the player that has a winning strategy determines the result of the model checking problem. The model checking game induces a game graph, which is used to determine which player has a winning strategy. The game graph can be computed on the fly, limited to its reachable states. This avoids exploring the parts of the model that are irrelevant for the formula to be checked. Hence, it addresses the issue raised in the work on *local* model checking.

A different characterization of the model checking problem for the alternation-free μ-calculus and in particular for CTL can be given in terms of so-called *1-letter-simple-weak alternating Buchi automata* (1SWABA), as part of the automata-theoretic approach to model checking [4, 29]. This approach derives optimal model checking algorithms for branching temporal logics using *alternating tree automata*. In general, the basic idea behind the automata-theoretic approach to model checking of branching time logics is to construct an alternating tree automaton such that its language is the set of all trees that satisfy the formula, i.e. it "describes" all the models that satisfy the given formula. Alternating tree automata generalize the standard notion of nondeterministic tree automata by allowing several successor states to go down along the same branch of the tree. Alternation is used to reduce the size of the automaton describing the formula from exponential in the length of the formula to linear in its length. The alternating automaton is assembled with the given model, resulting in the product automaton. The model checking problem is then reduced to the problem of checking nonemptiness of the language of the product automaton. The crucial observation is that the product automaton is a 1SWABA, thus for model checking it suffices to test the nonemptiness of the language of a 1SWABA, which is substantially simpler than solving the nonemptiness problem of tree automata.

The game-based approach to model checking, used in our work, is closely related to the automata-theoretic approach. The resemblance between these two approaches is described in [36], where it is shown how 1SWABA can be interpreted in terms of games, such that runs of a 1SWABA correspond to plays of a corresponding game from $\exists$loise's point of view. They define *co-runs* representing $\forall$belard's point of view and show that the 1SWABA has an accepting run iff $\exists$loise has a winning strategy and that the 1SWABA has an accepting co-run iff $\forall$belard has a winning strategy for the corresponding game. Furthermore, they rephrase the algorithm for checking nonemptiness of 1SWABA from [4] and show that it can be used to determine a winning strategy for the winner of the game. Thus, our work can also be described

in this framework, using alternating automata.

[19] also presents a local model checking algorithm for the alternation-free modal μ-calculus that is similar to the algorithm that results from the game-based or the automata-theoretic approach.

These model checking algorithms are all designed for concrete models. In our work we extend the discussion to abstract models and develop a game-based CTL model checking algorithm for them, as well as a refinement mechanism that is based on the properties of the game-based (or the automata-theoretic) model checking. We also exploit this approach in order to derive counterexamples for full CTL.

Abstraction

Model checking of realistic systems encounters the *state explosion problem*. One solution to it is the application of *abstraction* techniques, which aim to abstract the model to a smaller one, preserving formulae of some logic. Various abstraction techniques are formalized in the framework of *Abstract interpretation* [15, 38, 16].

[30, 3, 12, 8] discuss abstractions based on visible variables, where some of the system variables become *invisible*. These variables are treated as inputs, meaning that their behavior is non-deterministic.

[9] defines an abstraction based on variable clusters, which is more general than the invisible variables abstraction since it exploits logical relationships among variables. Their technique is similar to *predicate abstraction* (also called *boolean abstractions*) [23, 17, 44, 40, 45]. In predicate abstraction, abstract models are constructed by using boolean variables to represent concrete predicates. More specifically, [23] describes a method for the automatic construction of an abstract model using predicate abstraction, based on abstract interpretation. They consider a particular set of abstract states, which is the set of monomials on a set of state predicates. The successors of an abstract state are computed using the PVS theorem prover and upper approximations are constructed when needed. Thus, they allow verification of any universal temporal logic formula (without existential quantifiers). [17] has implemented a prototype system for efficient verification of invariants by predicate abstraction, based on the scheme presented in [23]. However, they use BDDs instead of monomials to represent the abstract state space, and the computation of successors is more accurate. [44] proposes an efficient algorithm for the automatic construction of boolean abstractions that requires fewer calls to decision procedures. [40] presents an algorithm that constructs a finite state abstract program from a concrete program by means of syntactic program transformations. They start with an initial set of predicates from a specification and iteratively compute the predicates required for the abstraction relative to the specification. All these works use the general framework of *existential abstraction* [14] and are thus suitable for verifying universal properties only (without existential quantifiers).

Unlike them, [45] shows how boolean abstractions can be constructed simply, efficiently and precisely while preserving properties in the full μ-calculus. They also

propose an automatic algorithm that is given a set of new predicates and refines the abstraction accordingly. The latter is based on the work of [16] that extends abstract interpretation to the analysis of both existential and universal properties, as expressible in the modal μ-calculus. They investigated how to define abstract models, based on abstract interpretation, such that the modal μ-calculus is preserved. Their approach can be applied to any abstraction within the framework of abstract interpretation. This implies that our work which considers CTL can be combined with any of these abstractions.

3-Valued Logic

Unlike the traditional (2-valued) abstraction, that preserves only truth of a formula from the abstract model to the concrete one, recently [6, 7, 20, 21, 28, 22] it was shown how automatic abstraction can be performed to verify modal μ-calculus formulae, based on a *3-valued semantics*, such that both truth and falseness are preserved. The key to make this possible is to present abstract systems using richer models that distinguish properties that are true, false and unknown of the concrete system. Different formalisms of abstract models suitable for the 3-valued semantics are proposed in the literature: Modal Transition Systems [33, 34], Partial Kripke Structures [6, 7], and Kripke Modal Transition Systems [28, 21]. It is shown in [22] that they have the same expressiveness and that their model checking problem can be reduced to two instances of traditional (2-valued) model checking. Thus, for any 3-valued formalism, 3-valued model checking has the same time and space complexity both in the size of the formula and the model as traditional 2-valued model checking. Such results were introduced in [20] and [27] for modal μ-calculus. In our work we use *Kripke Modal Transition Systems* [28, 21] and solve the model checking problem *directly*, without reducing it to traditional model checking. The direct solution has the same complexity as traditional model checking and it becomes helpful when refinement is needed.

Reasoning about such richer models requires 3-valued temporal logics [6]. As an enhancement of the standard 3-valued semantics, [7, 21] introduce the *thorough* 3-valued semantics. The thorough semantics gives more definite answers than the standard 3-valued semantics, at the expense of increasing the complexity of model checking: Interpreting a formula according to the thorough semantics is equivalent to solving two instances of *generalized model checking*. [7, 21] present algorithms and complexity bounds for the generalized model checking problem for various temporal logics and show that for propositional modal logic, CTL, or any branching time logic including CTL* or the modal μ-calculus, the generalized model checking problem has the same complexity as satisfiability, which is higher than the complexity of traditional model checking. In our work we use the standard 3-valued semantics [6], which is less precise, but enjoys a better complexity of the model checking algorithm, namely our algorithm has linear running time both in the size of the model and in the size of the formula.

It is shown in [20] that building a 3-valued abstraction, that can be used for model

checking any formula of the modal μ-calculus, can be done using existing abstraction techniques at the same computational cost as building a conservative 2-valued abstraction. They adapt existing predicate and cartesian abstraction techniques to get an abstraction that is monotonic, in the sense that adding predicates can only improve its precision both state-wise and transition-wise (usual predicate abstraction of Modal Transition Systems is not monotonic). Cartesian abstraction has no significant cost overhead and is compatible with the standard incremental refinement process for adding more predicates. Such abstractions can be used within our framework as well.

[49] also uses an abstraction mechanism based on 3-valued logic. They verify LTL properties of programs with dynamic allocation of objects (including thread objects) and references to objects. Their approach is different since it describes the model using first order logic. In addition, it deals with LTL, whereas our work is aimed at CTL properties.

Abstraction-Refinement

Kurshan [30] introduced *Localization reduction* with counterexample-guided refinement (also called *iterative abstraction-refinement*) for checking universal properties. This is an iterative technique that starts with an abstraction of the model and tries to verify the specification on this abstraction. If a counterexample is found a *reconstruction* process is executed to determine if it is a valid one. If the (abstract) counterexample is found to be *spurious*, the abstract model is refined to eliminate the possibility of this counterexample in the next iteration. The reduction (abstraction) used in their work is based on invisible variables. A similar approach is described in [2]. Other researchers [3, 12, 8] have also addressed localization reduction based on invisible variables. [3] presents algorithmic improvements to the localization reduction. They present a symbolic algorithm for path reconstruction including incremental refinement and backtracking. [12, 8] use SAT solvers in the counterexample analysis of AGp properties. [12] checks whether a counterexample is real or spurious with a SAT checker. They use a combination of Integer Linear Programming and machine learning techniques for refining the abstraction based on the counterexample. In [8] the abstract counterexamples obtained from model-checking the abstract model are symbolically simulated on the concrete system using a SAT checker. If no concrete counterexample is found, a subset of the invisible variables is reintroduced into the system. They introduce two algorithms for identifying the relevant variables to be reintroduced. These algorithms monitor the SAT checking phase in order to analyze the impact of individual variables.

[9] introduces *iterative abstraction-refinement* which is similar to the localization reduction. They use a more general abstraction based on variable clusters. [13] extends the work in [9], by generalizing the notion of counterexamples, and thus making this framework applicable to all ACTL properties.

[44] proposes an abstraction refinement framework for universal properties, using predicate abstraction. They propose to use the error traces generated by model

checking to automatically refine the abstraction. The refinement algorithm generates new predicates that will be used to enrich the abstract state-space.

Localization reduction [30], or iterative abstraction-refinement [9] provides a framework for model checking of universal properties, with counterexample guided refinement. In our work we consider abstraction-refinement for full CTL and do not restrict the discussion to universal properties. Other researchers have suggested abstraction-refinement mechanisms for various branching time temporal logics.

In [35], the tearing paradigm is presented as a way to automatically abstract behavior to obtain upper and lower bound approximations of a system. They present algorithms that exploit the bounds to perform conservative ACTL or ECTL model checking. Furthermore, an algorithm for false negative or false positive resolution is suggested based on the theory of a lattice of approximations, resulting in increasingly better approximations. Yet, their technique is restricted to ACTL or ECTL. In [41, 42] the full propositional μ-calculus is considered. In their abstraction, the concrete and abstract systems share the same state space. The simplification is based on taking supersets and subsets of a given set with a more compact BDD representation (to get under or over approximations). Refinement is based on a "goal set" of states which require further resolution. In [37] full CTL is handled. Their approximation techniques enable them to avoid rechecking the entire model after each refinement step while guaranteeing completeness. However, the verified system has to be described as a cartesian product of machines. The initial abstraction considers only machines that directly influence the formula and in each iteration the cone of influence is extended in a BFS manner. In each iteration they compute both an upper and a lower approximation to the states that satisfy the formula. [1] handles ACTL and full CTL. Their abstraction collapses all states that satisfy the same subformulae of φ into an abstract state. Thus, computing the abstract model is at least as hard as model checking. Instead, they use partial knowledge on the abstraction function and gain information in each refinement.

Our approach for abstraction-refinement is designed for full CTL and is applicable to any abstraction that can be described in the framework of abstract interpretation, thus it is more general. It also has the advantage of being most suitable for using results from previous iterations, resulting in an incremental algorithm.

Incremental Abstraction-Refinement

[26] introduces the concept of *lazy abstraction* to integrate and optimize the three phases of the abstract-check-refine loop within the abstraction-refinement framework. Lazy abstraction continuously builds and refines a single abstract model on demand, driven by the model checker, so that different parts of the model may exhibit different degrees of precision. Predicate abstraction is used and predicates are added only where necessary. The result is a nonuniform abstract model whose predicates change from state to state. They present an algorithm for model checking *safety* properties using lazy abstraction. The idea of lazy abstraction is also used in [25]. Our incremental algorithm generalizes the idea of Lazy abstraction to model checking of CTL

properties, where any abstraction that is described within the framework of abstract interpretation can be used.

Counterexamples and Deductive Proofs

We consider the issue of producing counterexamples in case the model refutes the examined property. A counterexample may be viewed as a proof of satisfaction for the *negation* of the property. This makes counterexamples and proofs closely related.

[43, 18, 39, 48] investigated the idea of generating temporal proofs from the information gained by model checking, when the verification succeeds. [18, 43] generate fully deductive proofs for LTL properties, when the verification is done using generalized Buchi automata or just discrete systems ([18]). They use the information in the product graph to generate a proof. [39] develops a deductive proof system for verifying branching time properties expressed in the μ-calculus and show how to generate a proof in this system from a model checking run. They use model checking that is based on parity games or alternating automata. The proof can be used to detect errors in a model checker and can allow integration of model checking with theorem proving. [48] introduces *support sets* which are abstract encodings of the "evidence" a model checker uses to justify its answer. They show how model checkers can be modified to compute support sets and how support sets can be used for generation of diagnostic information for explaining negative results and certifying the results of model checking (internal consistency).

Our approach is similar to these works in the sense that we use the information gained during the run of a model checker. However, we use it to present a counterexample, which is an extended sub-model, rather than a deductive proof. In this sense, our approach is closer to [13, 24]. [13] introduces tree-like counterexamples, which are a general form of ACTL counterexamples (and in fact suitable for a universal fragment of an extended branching time logic based on ω-regular temporal operators). They also present symbolic algorithms to generate tree-like counterexamples for ACTL specifications. Our work considers full CTL and is not limited to universal properties. This is also the case in [24], where counterexamples are annotated with additional proof steps. They develop a proof system and use a model checker as a decision procedure for construction of a proof. Yet, they provide only limited information in the case of counterexamples for ECTL and properties that use both universal and existential quantifiers, since proof "obligations" are used. In addition, unlike [24] that uses a model checker as a decision procedure in each step of the counterexample (proof) generation, we use the information gained in a *single* run of the model checker and produce a counterexample that is minimal and sufficient to explain refutation of any CTL formula.

1.2 Organization

The rest of the thesis is organized as follows. In the next chapter we give the necessary background for the game-based CTL model checking, abstractions and the 3-valued semantics. Due to technical reasons, we then start with the description of an annotated counterexample. Thus, in Chapter 3 we describe how to construct an annotated counterexample for full CTL and show that it is sufficient and minimal. In Chapter 4 we extend the game-based model checking algorithm to abstract models, using the 3-valued semantics. We then describe how to produce a concrete annotated counterexample using the abstract information in Chapter 5. In Chapter 6 we present our refinement technique, leading to an incremental abstraction-refinement framework, which is described in Chapter 7. Finally, we discuss some conclusions in Chapter 8.

Chapter 2

Preliminaries

Let AP be a finite set of atomic propositions. We define the set Lit of literals over AP to be the set $Lit = AP \cup \{\neg p : p \in AP\}$, i.e. for each $p \in AP$, both p and $\neg p$ are in Lit. We identify $\neg\neg p$ with p.

Definition 2.1 *The Logic CTL (Computation Tree Logic) in negation normal form is the set of formulae defined as follows:*

$$\varphi ::= true \mid false \mid l \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid A\psi \mid E\psi$$

where l ranges over Lit, and ψ is defined by

$$\psi ::= X\varphi \mid \varphi U \varphi \mid \varphi V \varphi$$

The (concrete) semantics of CTL formulae is defined with respect to a Kripke structure.

Definition 2.2 *A* Kripke Structure *is a tuple $M = (S, S_0, \rightarrow, L)$, where S is a finite set of states, $S_0 \subseteq S$ is a set of initial states, $\rightarrow \subseteq S \times S$ is a transition relation, which must be* total *(i.e., for every state $s \in S$ there exists a state $s' \in S$ such that $s \rightarrow s'$) and $L : S \rightarrow 2^{Lit}$ is a labeling function that associates each state in S with a subset of literals, such that for each state s and atomic proposition $p \in AP$, we have that exactly one of p and $\neg p$ is in $L(s)$, i.e. $p \in L(s)$ iff $\neg p \notin L(s)$.*

A path *in M is an infinite sequence of states, $\pi = s_0, s_1, \ldots$ such that for every $i \geq 0$, $s_i \rightarrow s_{i+1}$. If $s = s_0$, then π is said to be* from *s.*

$[(M, s) \models \varphi] = \mathrm{tt}$ means that the CTL formula φ is true in the state s of a Kripke structure M. $[(M, s) \models \varphi] = \mathrm{ff}$ means that φ is false in s. The formal definition follows.

Definition 2.3 *[11] The truth value* $\in \{tt, ff\}$ *of a CTL formula* φ *in a state* s *of a Kripke structure* $M = (S, S_0, \rightarrow, L)$*, denoted* $[(M, s) \models \varphi]$*, is defined inductively as follows:*

$$
\begin{array}{lcl}
[(M,s) \models true] & = & tt \\
[(M,s) \models false] & = & ff \\
[(M,s) \models l] & = & tt \;\Leftrightarrow\; l \in L(s), \text{ where } l \in Lit \\
[(M,s) \models \varphi_1 \wedge \varphi_2] & = & [(M,s) \models \varphi_1] \wedge [(M,s) \models \varphi_2] \\
[(M,s) \models \varphi_1 \vee \varphi_2] & = & [(M,s) \models \varphi_1] \vee [(M,s) \models \varphi_2] \\
[(M,s) \models A\psi] & = & tt \;\Leftrightarrow\; \forall \pi \text{ from } s : [(M,\pi) \models \psi] = tt \\
[(M,s) \models E\psi] & = & tt \;\Leftrightarrow\; \exists \pi \text{ from } s : [(M,\pi) \models \psi] = tt
\end{array}
$$

For a path $\pi = s_0, s_1, \ldots,$ $[(M, \pi) \models \psi]$ *is defined as follows.*

$$
\begin{array}{lcl}
[(M,\pi) \models X\varphi] & = & [(M,s_1) \models \varphi] \\
[(M,\pi) \models \varphi_1 U \varphi_2] & = & tt \;\Leftrightarrow\; \exists k \geq 0 : [([(M,s_k) \models \varphi_2] = tt) \\
 & & \qquad \wedge\; (\forall j < k : [(M,s_j) \models \varphi_1] = tt)] \\
[(M,\pi) \models \varphi_1 V \varphi_2] & = & tt \;\Leftrightarrow\; \forall k \geq 0 : [(\forall j < k : [(M,s_j) \models \varphi_1] = ff) \\
 & & \qquad \Rightarrow ([(M,s_k) \models \varphi_2] = tt)]
\end{array}
$$

We say that M *satisfies* φ*, denoted* $[M \models \varphi] = tt$*, if* $\forall s_0 \in S_0 : [(M, s_0) \models \varphi] = tt$*. Otherwise,* M *refutes* φ*, denoted* $[M \models \varphi] = ff$*.*

When M is clear from the context, we omit it from the notation and write $[s \models \varphi]$ or $[\pi \models \psi]$.

Definition 2.4 *Given a CTL formula* φ *of the form* $A(\varphi_1 U \varphi_2)$*,* $E(\varphi_1 U \varphi_2)$*,* $A(\varphi_1 V \varphi_2)$ *or* $E(\varphi_1 V \varphi_2)$*, its* expansion $exp(\varphi)$ *is defined as:*

$$
\begin{array}{lcl}
\varphi = A(\varphi_1 U \varphi_2) & : & exp(\varphi) = \{\varphi,\ \varphi_2 \vee (\varphi_1 \wedge AX\varphi),\ \varphi_1 \wedge AX\varphi,\ AX\varphi\} \\
\varphi = E(\varphi_1 U \varphi_2) & : & exp(\varphi) = \{\varphi,\ \varphi_2 \vee (\varphi_1 \wedge EX\varphi),\ \varphi_1 \wedge EX\varphi,\ EX\varphi\} \\
\varphi = A(\varphi_1 V \varphi_2) & : & exp(\varphi) = \{\varphi,\ \varphi_2 \wedge (\varphi_1 \vee AX\varphi),\ \varphi_1 \vee AX\varphi,\ AX\varphi\} \\
\varphi = E(\varphi_1 V \varphi_2) & : & exp(\varphi) = \{\varphi,\ \varphi_2 \wedge (\varphi_1 \vee EX\varphi),\ \varphi_1 \vee EX\varphi,\ EX\varphi\}
\end{array}
$$

2.1 Game-based Model Checking Algorithm

In this section we present the Game-theoretic approach to Model Checking of CTL formulae in a (concrete) Kripke structure [47, 36]. Given a Kripke structure $M = (S, S_0, \rightarrow, L)$ and a CTL formula φ, the *model checking game* of M and φ is defined as follows. Its board is the Cartesian product $S \times sub(\varphi)$ of the set of states S and the set of subformulae $sub(\varphi)$, where $sub(\varphi)$ is defined by:
if $\varphi =$ true, false or l where $l \in Lit$ then $sub(\varphi) = \{\varphi\}$.
if $\varphi = \varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$ then $sub(\varphi) = \{\varphi\} \cup sub(\varphi_1) \cup sub(\varphi_2)$.
if $\varphi = AX\varphi_1$ or $EX\varphi_1$ then $sub(\varphi) = \{\varphi\} \cup sub(\varphi_1)$.
if $\varphi = A(\varphi_1 U \varphi_2)$, $E(\varphi_1 U \varphi_2)$, $A(\varphi_1 V \varphi_2)$ or $E(\varphi_1 V \varphi_2)$ then $sub(\varphi) = exp(\varphi) \cup sub(\varphi_1) \cup sub(\varphi_2)$.

Given a state $s \in S$, the model checking game is played by two players, $\forall$belard, the refuter who wants to show that $[(M,s) \models \varphi] = \text{ff}$, and $\exists$loise, the prover who wants to show that $[(M,s) \models \varphi] = \text{tt}$. A single *play* from (s,φ) is a (possibly infinite) sequence $C_0 \rightarrow_{p0} C_1 \rightarrow_{p1} C_2 \rightarrow_{p2} \ldots$ of configurations, where $C_0 = (s,\varphi)$, $C_i \in S \times sub(\varphi)$ and $p_i \in \{\forall\text{belard}, \exists\text{loise}\}$. The subformula in C_i determines which player p_i makes the next move.

The possible moves at each step are:

1. $C_i = (s, \text{false})$, $C_i = (s, \text{true})$, or $C_i = (s,l)$ where $l \in Lit$: the play is finished. Such configurations are called *terminal configurations*.
2. $C_i = (s, AX\varphi)$: $\forall$belard chooses a transition $s \rightarrow s'$ in M and $C_{i+1} = (s', \varphi)$.
3. $C_i = (s, EX\varphi)$: $\exists$loise chooses a transition $s \rightarrow s'$ in M and $C_{i+1} = (s', \varphi)$.
4. $C_i = (s, \varphi_1 \wedge \varphi_2)$: $\forall$belard chooses $j \in \{1,2\}$ and $C_{i+1} = (s, \varphi_j)$.
5. $C_i = (s, \varphi_1 \vee \varphi_2)$: $\exists$loise chooses $j \in \{1,2\}$ and $C_{i+1} = (s, \varphi_j)$.
6. $C_i = (s, A(\varphi_1 U \varphi_2))$: $C_{i+1} = (s, \varphi_2 \vee (\varphi_1 \wedge AXA(\varphi_1 U \varphi_2)))$.
7. $C_i = (s, E(\varphi_1 U \varphi_2))$: $C_{i+1} = (s, \varphi_2 \vee (\varphi_1 \wedge EXE(\varphi_1 U \varphi_2)))$.
8. $C_i = (s, A(\varphi_1 V \varphi_2))$: $C_{i+1} = (s, \varphi_2 \wedge (\varphi_1 \vee AXA(\varphi_1 V \varphi_2)))$.
9. $C_i = (s, E(\varphi_1 V \varphi_2))$: $C_{i+1} = (s, \varphi_2 \wedge (\varphi_1 \vee EXE(\varphi_1 V \varphi_2)))$.

In configurations 6-9 the move is deterministic, thus any player can make the move.

A play is *maximal* iff it is infinite or ends in a terminal configuration. In [47] it has been shown that a play is infinite iff there is exactly one subformula of the form AU, EU, AV or EV that occurs infinitely often in the play. Such a subformula is called a *witness*.

Winning Criteria: $\forall$belard wins a (maximal) play iff one of the following holds:

1. the play is finite and ends in a terminal configuration of the form $C_i = (s, \text{false})$, or $C_i = (s,l)$, where $l \notin L(s)$.
2. the play is infinite and the witness is of the form AU or EU.

$\exists$loise wins the (maximal) play otherwise, i.e. iff one of the following holds:

1. the play is finite and ends in a terminal configuration of the form $C_i = (s, \text{true})$, or $C_i = (s,l)$, where $l \in L(s)$.
2. the play is infinite and the witness is of the form AV or EV.

The model checking *game* from (s, φ) consists of all the possible plays from (s, φ). A *strategy* is a set of rules for a player, telling him how to move in the current configuration. A *winning strategy* from (s, φ) is a set of rules allowing the player to win every play starting at (s, φ) if he plays by the rules. The following theorem tells us that the model checking problem can be reduced to the problem of finding which player has a winning strategy in the model checking game.

Theorem 2.5 *[47] Let M be a Kripke structure and φ a CTL formula. Then, for each $s \in S$:*

1. $[(M, s) \models \varphi] = tt$ *iff* $\exists$*loise has a winning strategy starting at* (s, φ).
2. $[(M, s) \models \varphi] = ff$ *iff* $\forall$*belard has a winning strategy starting at* (s, φ).

The model checking algorithm for the evaluation of $[M \models \varphi]$ consists of two parts. First, it constructs (part of) the *game-graph*. The *game-graph* is the graph whose nodes are the elements (configurations) of the game board and whose edges are the possible moves of the players. It captures all the possible plays of a game (from any configuration). The evaluation of the truth value of φ in M is then done in the second phase of the algorithm by coloring the game-graph.

2.1.1 Game-Graph Construction and its Properties

The truth value of φ in M depends on its truth value in the initial states of M. Thus, we are interested in plays that start from configurations in $S_0 \times \{\varphi\}$, referred to as *initial configurations*. The subgraph of the game-graph that is reachable from the *initial configurations* $S_0 \times \{\varphi\}$ is constructed in a BFS or DFS manner. The construction starts from the initial configurations (nodes) and applies each possible move, by the previously described rules, to get the successors in the game-graph of each new node. The result is denoted $G_{M \times \varphi} = (N, E)$, where $N \subseteq S \times sub(\varphi)$. The nodes (configurations) of the game-graph can be classified into three types.

1. Terminal configurations are leaves in the game-graph.
2. Nodes whose formula is of the form $\varphi_1 \wedge \varphi_2$ or $AX\varphi_1$ are $\wedge$*-nodes*.
3. Nodes whose formula is of the form $\varphi_1 \vee \varphi_2$ or $EX\varphi_1$ are $\vee$*-nodes*.

Nodes whose formula is of the form AU, EU, AV, EV can be considered either $\vee$-nodes or $\wedge$-nodes. Sometimes we further distinguish between nodes whose formula is of the form $AX\varphi$ ($EX\varphi$) and other $\wedge$-nodes ($\vee$-nodes) by referring to them as *AX-nodes* (*EX-nodes*). The edges in the game-graph are also divided to two types.

- Edges that originate in AX-nodes or EX-nodes are *progress* edges that reflect real transitions of the Kripke structure.

- Other edges are *auxiliary* edges.

An important property of the game-graph is described by the following lemma.

Lemma 2.6 *Let B be a non trivial strongly connected component (SCC) in a game-graph (a non-trivial SCC contains at least one edge). Then the set of formulae that are associated with the nodes in B is exactly one of the sets* $\exp(\varphi)$*, where* $\varphi \in \{A(\varphi_1 U \varphi_2),\ E(\varphi_1 U \varphi_2),\ A(\varphi_1 V \varphi_2),\ E(\varphi_1 V \varphi_2)\}$.

Proof: By the rules of the game, which determine the edges in the game-graph, we have that the sons of a node $n = (s, \varphi)$ in the game-graph are either associated with strict subformulae of φ, or with expansions of it in case φ is an AU, AV, EU, EV formula. Therefore, a non-trivial SCC B, which contains cycles, must have at least one node n in it that is associated with an AU, AV, EU, EV formula (otherwise we have a loop where each formula is a strict subformula of the previous one, which is impossible). Consider the case where the node n has the formula $\varphi = A(\varphi_1 U \varphi_2)$. Other cases are similar. We prove that in this case the set of formulae that are associated with the nodes in B is exactly $exp(A(\varphi_1 U \varphi_2))$.

First, we prove that B cannot contain additional formulae. Let $n' = (s', \varphi')$ be a node in B, other than n. We show that $\varphi' \in exp(A(\varphi_1 U \varphi_2))$. By the rules of the game, we have that the descendents of a node $n'' = (s'', \varphi'')$ in the game-graph are associated with subformulae from $sub(\varphi'')$. Since n' lies on the same SCC as n, we have that n' is a descendent of n, and thus $\varphi' \in sub(\varphi)$. Since $\varphi = A(\varphi_1 U \varphi_2)$, we have that $sub(\varphi) = exp(A(\varphi_1 U \varphi_2)) \cup sub(\varphi_1) \cup sub(\varphi_2)$. It remains to show that $\varphi' \notin sub(\varphi_1) \cup sub(\varphi_2)$, thus it must be the case that $\varphi' \in exp(A(\varphi_1 U \varphi_2))$. Suppose the contrary, i.e. $\varphi' \in sub(\varphi_1) \cup sub(\varphi_2)$. This implies that $sub(\varphi') \subseteq sub(\varphi_1) \cup sub(\varphi_2)$. Since n is also a descendent of n', we have that $\varphi \in sub(\varphi')$. i.e. $\varphi \in sub(\varphi_1) \cup sub(\varphi_2)$. Thus φ must obey one of the following.

1. $|\varphi| \leq |\varphi_1|$ or $|\varphi| \leq |\varphi_2|$, or:
2. φ is of the form $(\varphi_1' \vee \varphi_2')$ or $(\varphi_1' \wedge \varphi_2')$ or $AX\varphi'$ (results from expansion).

Obviously, $\varphi = A(\varphi_1 U \varphi_2)$ obeys none of the above. Contradiction.

To complete the proof, it remains to show that B must contain *all* the formulae in $exp(A(\varphi_1 U \varphi_2))$. This is clear from the structure of the game-graph: if one of these formulae were missing, no loop could be formed in contradiction to the fact that B is a non-trivial SCC. □

Based on Lemma 2.6, we generalize the notion of a witness in the context of the game-graph. The formula φ such that $exp(\varphi)$ is the set of formulae in a non-trivial SCC is called a *witness*. Each non-trivial SCC is classified as an AU, AV, EU, or EV SCC, based on its witness.

2.1.2 Coloring Algorithm

The following *Coloring Algorithm* [5] labels each node (configuration) in the game-graph $G_{M\times\varphi}$ by T or F, depending on whether $\exists$loise or $\forall$belard has a winning strategy for the game that starts at that node.

The game-graph is partitioned into its *Maximal Strongly Connected Components* (MSCCs), denoted Q_i's, and an order $\leq$ is determined on the Q_i's, such that an edge (n, n'), where $n \in Q_i$ and $n' \in Q_j$, exists in the game-graph only if $Q_j \leq Q_i$. Such an order exists because the MSCCs of the game-graph form a *directed acyclic graph* (DAG). It can be extended to a total order $\leq$ arbitrarily.

The coloring algorithm processes the Q_i's according to the determined order, bottom-up. Let Q_i be the smallest MSCC with respect to $\leq$ that is not yet fully colored. Hence, every outgoing edge of a node in Q_i leads either to a colored node or to a node in the same set, Q_i. The nodes of Q_i are colored as follows.

1. Terminal nodes in Q_i are colored by T if $\exists$loise wins in them, and by F otherwise.
2. An $\vee$-node is colored by T if it has a son that is colored T, and by F if all its sons are colored F.
3. An $\wedge$-node is colored by T if all its sons are colored T, and by F if it has a son that is colored F.
4. All the nodes in Q_i that remain uncolored after the propagation of these rules are colored according to the witness in Q_i (by Lemma 2.6 there exists exactly one such witness). They are colored by F if the witness is of the form AU or EU, and are colored by T if the witness is of the form AV or EV.

The result of the coloring algorithm is a *coloring function* $\chi : N \to \{T, F\}$.

Theorem 2.7 *[47] Let $G_{M\times\varphi}$ be a game-graph and let n be a node in the game-graph, then:*

1. $\chi(n) = T$ *iff* $\exists$*loise has a winning strategy starting at* n.
2. $\chi(n) = F$ *iff* $\forall$*belard has a winning strategy starting at* n.

As a conclusion of Theorem 2.5 and Theorem 2.7, we get the following theorem.

Theorem 2.8 *[47] Let M be a Kripke structure and φ a CTL formula. Then, for each $n = (s, \varphi_1) \in G_{M\times\varphi}$:*

1. $[(M, s) \models \varphi_1] = tt$ *iff* $n = (s, \varphi_1)$ *is colored by* T.
2. $[(M, s) \models \varphi_1] = ff$ *iff* $n = (s, \varphi_1)$ *is colored by* F.

Based on Theorem 2.8 we conclude that if every initial configuration $n_0 \in S_0 \times \{\varphi\}$ is colored by T, then $[M \models \varphi] = \text{tt}$. Otherwise, $[M \models \varphi] = \text{ff}$.

2.2 Abstraction

In this section we present abstract models (for CTL) and their relation to concrete models. So far, we considered Kripke structures that represent concrete models, and discussed the semantics of CTL formulae with respect to them. However, concrete Kripke structures may be very large. Consequently, their model checking problem becomes infeasible due to the state explosion problem. A powerful solution is based on using abstractions of the concrete model.

It turns out that in order to guarantee preservation of CTL formulae from abstract models to concrete models, we need to introduce *two* transition relations [33, 16]: preservation of *universal* properties requires an over-approximation, whereas preservation of *existential* properties requires an under-approximation. This is accomplished by using *Kripke Modal Transition Systems* [28, 21].

Definition 2.9 *A* Kripke Modal Transition System *(KMTS) is a tuple* $M = (S, S_0, \xrightarrow{must}, \xrightarrow{may}, L)$*, where* S *is a finite set of states,* $S_0 \subseteq S$ *is a set of initial states,* $\xrightarrow{must} \subseteq S \times S$ *and* $\xrightarrow{may} \subseteq S \times S$ *are transition relations such that the relation* $\xrightarrow{may}$ *is* total *and* $\xrightarrow{must} \subseteq \xrightarrow{may}$*, and* $L : S \to 2^{Lit}$ *is a labeling function that associates each state in* S *with literals from* Lit*, such that for each state* s *and atomic proposition* $p \in AP$*,* at most *one of* p *and* $\neg p$ *is in* $L(s)$*.*

A must (may) *path in* M *is a* maximal *sequence of states,* $\pi = s_0, s_1, \ldots$ *such that for every two consecutive states* s_i, s_{i+1} *in* π*, we have that* $s_i \xrightarrow{must} s_{i+1}$ *(*$s_i \xrightarrow{may} s_{i+1}$*). The maximality is in the sense that* π *cannot be extended by any other transition of the same type. If* $s = s_0$*, then* π *is said to be* from s*.*

Transitions in $\xrightarrow{\text{must}}$ are called *must* transitions, and transitions in $\xrightarrow{\text{may}}$ are called *may* transitions. Note that since the transition relation $\xrightarrow{\text{may}}$ is total, then every may path is infinite (due to the maximality), whereas a (maximal) must path can be finite, since the transition relation $\xrightarrow{\text{must}}$ is not necessarily total. This means that although every must-transition is also a may-transition (by definition), the same does *not* necessarily hold for paths. That is, a finite must path is not considered a may path, because it is not maximal in terms of may transitions. Since we now consider finite paths in addition to infinite paths, we need the following definition.

Definition 2.10 *Let* π *be a must or may path. The* length *of* π*, denoted* $|\pi|$*, is defined to be the number of transitions in* π*. That is,*

$$|\pi| = \begin{cases} \infty & \text{if } \pi \text{ is infinite} \\ n & \text{if } \pi \text{ is finite and of the form } s_0, \ldots, s_n \end{cases}$$

Note, that a Kripke structure can be viewed as a KMTS where $\rightarrow = \xrightarrow{\text{must}} = \xrightarrow{\text{may}}$, and for each state s and atomic proposition $p \in AP$, we have that *exactly* one of p and $\neg p$ is in $L(s)$.

We consider abstractions that are done by collapsing sets of concrete states (from S_C) into single abstract states (in S_A). Such abstractions can be described in the framework of *Abstract Interpretation* [38, 16].

Definition 2.11 *[15, 38]* $(\alpha : C \to A, \gamma : A \to C)$ *is a* Galois connection *from* $(C, \preceq)$ *to* $(A, \sqsubseteq)$ *iff (1)* α *and* γ *are total and monotonic, (2) for all* $c \in C$, $\gamma \circ \alpha(c) \succeq c$, *and (3) for all* $a \in A$, $\alpha \circ \gamma(a) \sqsubseteq a$.

Let $M_C = (S_C, S_{0C}, \to, L_C)$ be a (concrete) Kripke structure. Let $(S_A, \sqsubseteq)$ be a poset of *abstract states* and $(\gamma : S_A \to 2^{S_C}, \alpha : 2^{S_C} \to S_A)$ a *Galois connection* from $(2^{S_C}, \subseteq)$ to $(S_A, \sqsubseteq)$, that determines its relation to the concrete states. γ is the *concretization function* that maps each abstract state to the set of concrete states that it represents. α is the *abstraction function* that maps each set of concrete states to the abstract state that represents it.

An abstract model $M_A = (S_A, S_{0A}, \xrightarrow{\text{must}}, \xrightarrow{\text{may}}, L_A)$ can then be defined as follows. The set of initial abstract states S_{0A} is built such that each concrete initial state is represented by an abstract initial state and there are no additional initial abstract states, i.e. $s_{0a} \in S_{0A}$ *iff* there exists $s_{0c} \in S_{0C}$ such that $s_{0c} \in \gamma(s_{0a})$. The requirement that there are no additional initial abstract states is needed to ensure preservation of falsity in the *model*, as described in the second part of Theorem 2.14. This requirement is not needed for state-wise preservation, described in the first part of the theorem.

The labeling of an abstract state is done according to the labeling of all the concrete states that it represents. An abstract state s_a is labeled by $l \in Lit$, only if all the concrete states that are represented by it are labeled by l as well. Therefore, it is possible that neither p nor $\neg p$ are in $L_A(s_a)$.

The *may*-transitions in an abstract model are computed such that every concrete transition between two states is represented by them: if $\exists s_c \in \gamma(s_a)$ and $\exists s'_c \in \gamma(s'_a)$ such that $s_c \to s'_c$, then there exists a may-transition $s_a \xrightarrow{\text{may}} s'_a$. Note that it is possible that there are additional may-transitions as well. The *must*-transitions, on the other hand, represent concrete transitions that are common to all the concrete states that are represented by the origin abstract state: a must-transition $s_a \xrightarrow{\text{must}} s'_a$ exists only if $\forall s_c \in \gamma(s_a)$ we have that $\exists s'_c \in \gamma(s'_a)$ such that $s_c \to s'_c$. Note that it is possible that there are less must-transitions than allowed by this rule. That is, the may and must transitions do not have to be *accurate*, as long as they maintain these conditions. Also note, that since the concrete transition relation is total, then the resulting abstract transition relation $\xrightarrow{\text{may}}$ is also total, as required. The abstract transition relation $\xrightarrow{\text{must}}$, on the other hand, is not necessarily total. In fact, it can even be empty.

Other constructions of abstract models, based on Galois connections, can be found in [16, 20].

The resulting abstract model is *more abstract* than M_C as defined by the following definition, which formalizes the relation between an abstract model and a concrete model that guarantees preservation of CTL formulae.

Definition 2.12 *[33, 16, 21] Let $M_C = (S_C, S_{0C}, \rightarrow, L_C)$ be a concrete Kripke structure, and let $M_A = (S_A, S_{0A}, \xrightarrow{must}, \xrightarrow{may}, L_A)$, be an abstract KMTS. We say that $H \subseteq S_C \times S_A$ is a* mixed simulation *from M_C to M_A if $(s_c, s_a) \in H$ implies the following:*

1. *$L_A(s_a) \subseteq L_C(s_c)$.*

2. *if $s_c \rightarrow s'_c$, then there is some $s'_a \in S_A$ such that $s_a \xrightarrow{may} s'_a$ and $(s'_c, s'_a) \in H$.*

3. *if $s_a \xrightarrow{must} s'_a$, then there is some $s'_c \in S_C$ such that $s_c \rightarrow s'_c$ and $(s'_c, s'_a) \in H$.*

If there exists a mixed simulation H such that for each $s_c \in S_{0C}$ there exists $s_a \in S_{0A}$ such that $(s_c, s_a) \in H$ and for each $s_a \in S_{0A}$ there exists $s_c \in S_{0C}$ such that $(s_c, s_a) \in H$, we say that M_A is more abstract *than M_C, denoted $M_C \preceq M_A$.*

The mixed simulation relation $H \subseteq S_C \times S_A$ from M_C to an abstract model which is constructed based on a Galois connection as described above is induced by the concretization function as follows. H is defined such that $(s_c, s_a) \in H$ iff $s_c \in \gamma(s_a)$. The results presented in this thesis are applicable to *any* abstract model that is *more abstract* than the concrete model M_C with respect to the mixed simulation relation, and are not limited to our construction of an abstract model.

[28] defines the *3-valued semantics* of a CTL formula over a KMTS. The 3-valued semantics is designed to be *conservative* in the sense that it preserves both satisfaction (tt) and refutation (ff) of a formula from the abstract model to the concrete one. However, a new truth value, $\perp$ is introduced. If the truth value of a formula in an abstract model is $\perp$, the meaning is that its value over the concrete model is not known and can be either tt or ff.

Definition 2.13 *The 3-valued semantics of a CTL formula φ in a state s of a KMTS $M = (S, S_0, \xrightarrow{must}, \xrightarrow{may}, L)$, denoted $[(M, s) \stackrel{3}{\models} \varphi]$, is defined inductively as follows:*

$$
\begin{array}{lcl}
[(M,s) \stackrel{3}{\models} true] & = & tt \\
[(M,s) \stackrel{3}{\models} false] & = & ff \\
[(M,s) \stackrel{3}{\models} l] & = & \begin{cases} tt & \text{if } l \in L(s) \\ ff & \text{if } \neg l \in L(s) \\ \bot & \text{otherwise} \end{cases} \\
[(M,s) \stackrel{3}{\models} \varphi_1 \wedge \varphi_2] & = & \begin{cases} tt & \text{if } [(M,s) \stackrel{3}{\models} \varphi_1] = tt \text{ and } [(M,s) \stackrel{3}{\models} \varphi_2] = tt \\ ff & \text{if } [(M,s) \stackrel{3}{\models} \varphi_1] = ff \text{ or } [(M,s) \stackrel{3}{\models} \varphi_2] = ff \\ \bot & \text{otherwise} \end{cases} \\
[(M,s) \stackrel{3}{\models} \varphi_1 \vee \varphi_2] & = & \begin{cases} tt & \text{if } [(M,s) \stackrel{3}{\models} \varphi_1] = tt \text{ or } [(M,s) \stackrel{3}{\models} \varphi_2] = tt \\ ff & \text{if } [(M,s) \stackrel{3}{\models} \varphi_1] = ff \text{ and } [(M,s) \stackrel{3}{\models} \varphi_2] = ff \\ \bot & \text{otherwise} \end{cases} \\
[(M,s) \stackrel{3}{\models} A\psi] & = & \begin{cases} tt & \text{if for each may-path } \pi \text{ from } s : [(M,\pi) \stackrel{3}{\models} \psi] = tt \\ ff & \text{if there exists a must-path } \pi \text{ from } s \text{ such that } : \\ & \qquad [(M,\pi) \stackrel{3}{\models} \psi] = ff \\ \bot & \text{otherwise} \end{cases} \\
[(M,s) \stackrel{3}{\models} E\psi] & = & \begin{cases} tt & \text{if there exists a must-path } \pi \text{ from } s \text{ such that } : \\ & \qquad [(M,\pi) \stackrel{3}{\models} \psi] = tt \\ ff & \text{if for each may-path } \pi \text{ from } s : [(M,\pi) \stackrel{3}{\models} \psi] = ff \\ \bot & \text{otherwise} \end{cases}
\end{array}
$$

For a may or must path $\pi = s_0, s_1, \ldots$, $[(M,\pi) \stackrel{3}{\models} \psi]$ *is defined as follows.*

$$
\begin{array}{lcl}
[(M,\pi) \stackrel{3}{\models} X\varphi] & = & \begin{cases} [(M,s_1) \stackrel{3}{\models} \varphi] & \text{if } |\pi| > 0 \\ \bot & \text{otherwise} \end{cases} \\
[(M,\pi) \stackrel{3}{\models} \varphi_1 U \varphi_2] & = & \begin{cases} tt & \text{if } \exists\, 0 \le k \le |\pi| : [([(M,s_k) \stackrel{3}{\models} \varphi_2] = tt) \\ & \qquad \wedge (\forall j < k : [(M,s_j) \stackrel{3}{\models} \varphi_1] = tt)] \\ ff & \text{if } \big(\forall\, 0 \le k \le |\pi| : [(\forall j < k : [(M,s_j) \stackrel{3}{\models} \varphi_1] \neq ff) \\ & \qquad \Rightarrow ([(M,s_k) \stackrel{3}{\models} \varphi_2] = ff)]\big) \\ & \quad \wedge \big((\forall\, 0 \le k \le |\pi| : [(M,s_k) \stackrel{3}{\models} \varphi_1] \neq ff) \Rightarrow |\pi| = \infty\big) \\ \bot & \text{otherwise} \end{cases} \\
[(M,\pi) \stackrel{3}{\models} \varphi_1 V \varphi_2] & = & \begin{cases} tt & \text{if } \big(\forall\, 0 \le k \le |\pi| : [(\forall j < k : [(M,s_j) \stackrel{3}{\models} \varphi_1] \neq tt) \\ & \qquad \Rightarrow ([(M,s_k) \stackrel{3}{\models} \varphi_2] = tt)]\big) \\ & \quad \wedge \big((\forall\, 0 \le k \le |\pi| : [(M,s_k) \stackrel{3}{\models} \varphi_1] \neq tt) \Rightarrow |\pi| = \infty\big) \\ ff & \text{if } \exists\, 0 \le k \le |\pi| : [([(M,s_k) \stackrel{3}{\models} \varphi_2] = ff) \\ & \qquad \wedge (\forall j < k : [(M,s_j) \stackrel{3}{\models} \varphi_1] = ff)] \\ \bot & \text{otherwise} \end{cases}
\end{array}
$$

We say that $[M \stackrel{3}{\models} \varphi] = tt$ *if* $\forall s_0 \in S_0 : [(M,s_0) \stackrel{3}{\models} \varphi] = tt$. *We say that* $[M \stackrel{3}{\models} \varphi] = ff$ *if* $\exists s_0 \in S_0 : [(M,s_0) \stackrel{3}{\models} \varphi] = ff$. *Otherwise,* $[M \stackrel{3}{\models} \varphi] = \bot$.

Intuitively, the 3-valued semantics is defined such that a formula is evaluated to tt or ff only when the abstract information suffices to determine such a definite truth

value that will hold in the represented concrete model. Therefore, truth of universal formulae (of the form $A\psi$) is examined along *all* the may paths (which represent at least all the concrete paths), whereas falsity of such formulae is shown by a *single* must path (which represents a definite concrete path), and dually for existential formulae (of the form $E\psi$). Similar arguments apply to the evaluation of path formulae of the form $X\varphi,\ \varphi_1 U \varphi_2,\ \varphi_1 V \varphi_2$. For example, in order to say that the truth value of $\varphi_1 U \varphi_2$ in a path is true, φ_2 needs to become true *within* the path. In order to say that $\varphi_1 U \varphi_2$ is false in the path we require that at every position, if φ_1 is not false yet, then φ_2 is still false (the eventuality is not fulfilled), and we also require that if φ_1 is *never* false, then the path is infinite. The latter requirement is needed because if we have a *finite* path where φ_1 is never false (and φ_2 is always false, based on the first requirement) then we cannot claim that the truth value of $\varphi_1 U \varphi_2$ is false. The reason is that real concrete paths are infinite, thus in the concrete model that is represented by the abstract KMTS there is still "hope" that φ_2 will become true in the future (and the eventuality will be fulfilled), although it is not reflected by the abstract path. Since φ_1 has not become false yet, this can make the until formula true in the concrete model. This leaves us with two possibilities for falsity of $\varphi_1 U \varphi_2$: either φ_1 becomes false *within* the path (when φ_2 is still false as well), or the path is infinite and φ_2 is false all along. Yet, if φ_2 is always false along a *finite* path, then this information alone is not sufficient in order to say that the until formula is false.

The preservation of CTL formulae from an abstract model to a concrete model is guaranteed by the following theorem.

Theorem 2.14 *[21] Let $H \subseteq S_C \times S_A$ be a mixed simulation relation from a Kripke structure M_C to a KMTS M_A. Then for every $(s_c, s_a) \in H$ and every CTL formula φ, we have that:*

1. $[(M_A, s_a) \stackrel{3}{\models} \varphi] = tt$ *implies that* $[(M_C, s_c) \models \varphi] = tt$.
2. $[(M_A, s_a) \stackrel{3}{\models} \varphi] = ff$ *implies that* $[(M_C, s_c) \models \varphi] = ff$.

We conclude that if $M_C \preceq M_A$, then for every CTL formula φ, we have that:

1. $[M_A \stackrel{3}{\models} \varphi] = tt$ *implies that* $[M_C \models \varphi] = tt$.
2. $[M_A \stackrel{3}{\models} \varphi] = ff$ *implies that* $[M_C \models \varphi] = ff$.

Chapter 3

Using Games to Produce Annotated Counterexamples

In this chapter we consider the concrete semantics of CTL. We refer to (concrete) Kripke structures and describe how to construct an *annotated counterexample* from a game-graph for M and φ, as well as the information gained by the coloring algorithm, in case M does not satisfy φ.

First, the coloring algorithm described in Chapter 2.1 is changed to identify and remember the *cause* of the coloring of an $\wedge$-node n that is colored by F. If n was colored by its sons, then $cause(n)$ is the son that was the first to be colored by F. If n was colored due to a witness, then $cause(n)$ is chosen to be one of its sons which resides on the same SCC and was colored by witness as well. There must exist such a son, otherwise n would be colored by its sons. Note that $cause(n)$ depends on the execution of the coloring algorithm.

Given a game-graph $G_{M\times\varphi}$, for a Kripke structure M and a CTL formula φ, and given its coloring χ and an initial node $n_0 = (s_0, \varphi)$ such that $\chi(n_0) = F$, the following DFS/BFS-like algorithm finds an *annotated counterexample* over the nodes of $G_{M\times\varphi}$. The computed annotated counterexample, denoted $C_{M\times\varphi}$ and in short C, is a subgraph of the given game-graph $G_{M\times\varphi}$, colored by F.

Algorithm `ComputeCounter`
Initially: $new = \{(s_0, \varphi)\}$, $C = \emptyset$.
while $new \neq \emptyset$
n = remove (new)

- if n was already handled - continue.
- if n is a terminal node - continue. $\backslash * \; sons = \emptyset \; * \backslash$
- if n is an $\vee$-node - for each son n' of n add n' to new and the edge (n, n') to C.
- if n is an $\wedge$-node - add $cause(n)$ to new and the edge $(n, cause(n))$ to C.

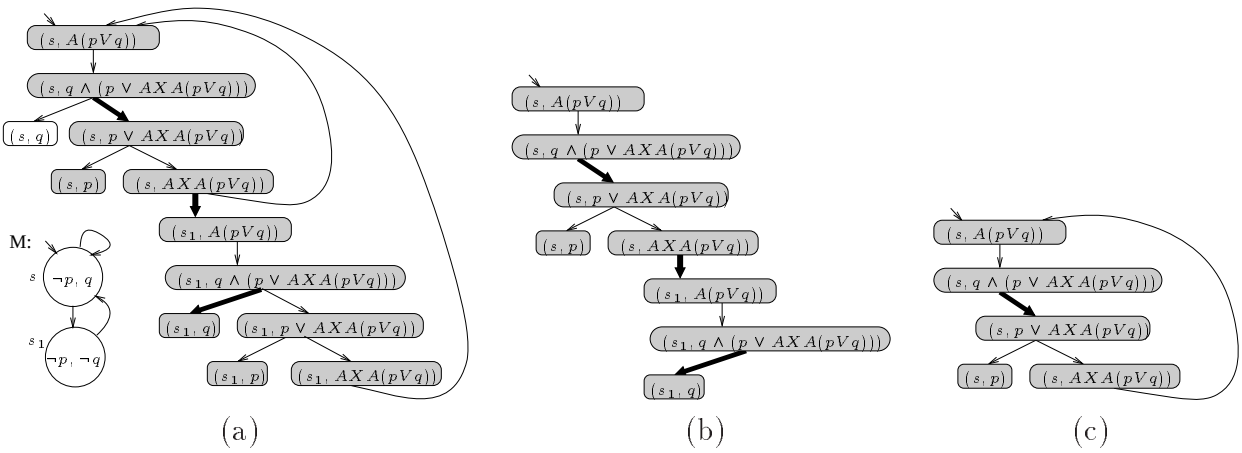

Figure 3.1: (a) A colored game-graph for M and $\varphi = A(pVq)$, where white nodes are colored by T, grey nodes are colored by F and bold edges point to the *cause* of an $\wedge$-node, (b) Its annotated counterexample computed by `ComputeCounter`, and (c) A possible result of `ComputeCounter` without the use of the *cause* in $\wedge$-nodes.

Note, that we construct C by adding edges to it, with the meaning that C consists of the corresponding nodes connected by these edges.

Complexity: Clearly, the construction of the annotated counterexample has a linear running time in the size of its result. The result is linear (in the worst case) with respect to the size of the game-graph $G_{M\times\varphi}$. The latter is bounded by the size of the underlying Kripke structure times the length of the CTL formula, i.e. $O(|M| \cdot |\varphi|)$.

The computed annotated counterexample can be viewed as the part of the winning strategy of the refuter that is sufficient to guarantee its victory. We formalize and prove this notion in the next section. Intuitively speaking, it is indeed a counterexample in the sense that it points out the reasons for φ's refutation on the model. Each node in the computed annotated counterexample C is marked by a state s and by a subformula φ_1, such that $\chi((s,\varphi_1)) = F$ (as claimed by Lemma 3.2), thus by Theorem 2.8, $[s \models \varphi_1] = \mathrm{ff}$. The edges point out the reason (cause) for the refutation of a certain subformula in a certain state: the refutation in an $\wedge$-node is shown by refutation in one of its sons, whereas the refutation in an $\vee$-node is shown by all its sons. Hence, by analyzing the computed annotated counterexample, one can understand why each subformula, and in particular the main formula, is refuted in the relevant state(s).

Note, that for the correctness of the algorithm `ComputeCounter` and its result, it is *mandatory* to choose for an $\wedge$-node the son that caused the coloring of the node, and not any son that was colored by F. The following example demonstrates the importance of the use of the *cause*.

Example 3.1 Figure 3.1 presents an example for computing an annotated counterexample using the algorithm `ComputeCounter` and demonstrates the necessity of choosing $cause(n)$ as a son of an $\wedge$-node n when computing an annotated counterexample. Figure 3.1(a) presents a colored game-graph G for the model M and $\varphi = A(pVq)$, where grey nodes are colored by F, whereas white nodes are colored by T, and bold edges point to the *cause* of an $\wedge$-node. The coloring algorithm that results in this coloring partitions G into five MSCCS: $Q_1 = \{(s,q)\}, Q_2 = \{(s,p)\}, Q_3 = \{(s_1,q)\}, Q_4 = \{(s_1,p)\}$ and Q_5 consists of the rest of the nodes. The sets $Q_1 - Q_4$, that have no outgoing edges, can be ordered arbitrarily amongst themselves, but they are all smaller than Q_5, since Q_5 has an outgoing edge to each of them. Thus the terminal nodes in $Q_1 - Q_4$ are colored before Q_5 is handled. When Q_5 is processed, $(s_1, q \wedge (p \vee AXA(pVq)))$ is colored F based on (s_1,q) (its *cause*). This causes $(s_1, A(pVq))$, $(s, AXA(pVq))$, $(s, p \vee AXA(pVq))$, $(s, q \wedge (p \vee AXA(pVq)))$, $(s, A(pVq))$, $(s_1, AXA(pVq))$ and $(s_1, p \vee AXA(pVq))$ (in this order) to be colored F as well. Thus, $(s_1, A(pVq))$ is the *cause* of $(s, AXA(pVq))$ and $(s, p \vee AXA(pVq))$ is the *cause* of $(s, q \wedge (p \vee AXA(pVq)))$. Furthermore, the initial node $(s, A(pVq))$ is colored by F, i.e. $[s \models A(pVq)] =$ ff. Figure 3.1(b) presents the annotated counterexample computed by `ComputeCounter`, where it can be seen that the reason for refutation is the existence of the path $s, s_1, \ldots$ and particularly its prefix s, s_1, where q is not satisfied by s_1, although it was not "released" by p (p does not hold in s). On the other hand, Figure 3.1(c) presents a subgraph of G, that is computed by a variation of `ComputeCounter`, where for an $\wedge$-node, an arbitrary son that is colored by F is chosen. In the example, the node $(s, A(pVq))$ was chosen as a refuting son of $(s, AXA(pVq))$ rather than $(s_1, A(pVq))$, which is its cause. The resulting subgraph implies that the refutation of $A(pVq)$ results from the path $s, s, \ldots$. However, this path satisfies pVq, such that it does not prove refutation. Thus, this is not a "good" counterexample. This will be formally shown in Chapter 3.2, where the notion of an annotated counterexample is formalized.

3.1 Properties of the Annotated Counterexample

The annotated counterexample produced by `ComputeCounter`, denoted C, is a subgraph of the game-graph, and as such it has the properties of the game-graph. In addition, it has the following properties.

Lemma 3.2 *For each node $n \in C$, we have that $\chi(n) = F$.*

Proof: By its construction, all the nodes in the computed annotated counterexample C are colored by F. This can be shown by induction on the construction of C, when we rely on the property of χ that an $\vee$-node is colored by F iff all its sons are colored by F and an $\wedge$-node is colored by F iff at least one of its sons is colored by F. This property is obviously correct when the coloring does not use a witness, but it is also true when a witness causes the coloring. $\Box$

Lemma 3.3 *C contains non-trivial SCCs* if and only if *at least one of the nodes in the SCC was colored due to a witness.*

Proof: Clearly, if the coloring of a node that appears in the computed annotated counterexample C was based on a witness, then this node resides on a non-trivial SCC that will be added to C. This results from the following properties. For an $\vee$-node all the sons are added to the annotated counterexample and in particular the one(s) in the non-trivial SCC. For an $\wedge$-node that is colored by a witness, its cause is added to the annotated counterexample, where the cause is a son within the non-trivial SCC that is also colored by a witness. Thus, a cycle is formed.

To prove the second implication, let us look at a non trivial SCC in C. All the nodes in it are colored by F. Assume that all of them were colored due to their sons. Consider the first node on the SCC that was colored and denote it by n_1. Since it is the first, it must be colored by F based on its sons *outside* the SCC. Yet, it obviously has sons within the SCC too. Thus it must be an $\wedge$-node. The reason for this conclusion is that only $\wedge$-nodes can be colored by F based on part of their sons only. However, an $\wedge$-node has exactly one son in C, and by construction this son is its *cause*, i.e. the node that caused its coloring, which is outside the SCC by our assumption. Thus, it is not possible that n_1 has another son in C within the SCC, which contradicts the fact that n_1 resides on the SCC. We conclude that at least one of the nodes in the SCC was colored due to a witness. $\square$

From Lemma 3.2 and Lemma 3.3 we have the following conclusion.

Corollary 3.4 *Non-trivial SCCs in C are either AU-SCCs or EU-SCCs.*

Proof: Lemma 3.3 tells us that if a non-trivial SCC appears in C then at least one of its nodes was colored by a witness. On the other hand, by Lemma 3.2 we know that all the nodes in C are colored by F, and by the coloring algorithm we know that only nodes in AU or EU SCCs are colored by F due to witness. Thus, the corollary is implied. $\square$

The property of C described in Lemma 3.3, along with Corollary 3.4, imply that non-trivial SCCs appear in C iff at least one of their nodes was colored due to an AU or EU witness. That is, any non-trivial SCC that appears in the annotated counterexample indicates a refutation of the U operator, which results, at least partly, from an infinite path, where weak until[1] is satisfied, but not strong until (which is used in our work). This intuition results from the properties of the coloring algorithm. If a node is colored due to a witness, this means that finite information alone is not sufficient to cause its color. In the case of $A(\varphi_1 U \varphi_2)$, this means that there is no finite (prefix of a) path where φ_1 ceases being satisfied before φ_2 is satisfied, and the refutation results from an infinite path where φ_1 is always satisfied, but φ_2 is never satisfied. In case of $E(\varphi_1 U \varphi_2)$, this means that the refutation results, at least partly, from infinite evidence of this form and not only from finite (prefixes of) paths.

[1] The weak version of the until operator, $\varphi_1 W \varphi_2$, does not guarantee that φ_2 holds eventually.

Since the algorithm `ComputeCounter` is designed to find counterexamples for full CTL, and in particular for existential properties, its result C has a more complex structure than counterexamples that are used for universal properties. Yet, the following Lemma shows that when applied to formulae in ACTL, where only universal properties exist, the result of `ComputeCounter` has a simpler structure. In fact, it has a *tree-like* structure, as defined in [13]. It differs from the counterexamples presented in [13] only in the existence of annotations.

Lemma 3.5 *Non-trivial AU-SCCs in C are always simple cycles, rather than general SCCs.*

Proof: Consider a non-trivial AU-SCC in C. By the construction of C, we have that $\wedge$-nodes in the SCC have a single son in C and in particular in the SCC. Apart from $\wedge$-nodes, such an SCC contains only $\vee$-nodes, that are not EX-nodes. This is because by Lemma 2.6, the game-graph, and C in particular, have the property that the set of formulae in a non-trivial AU-SCC is exactly $exp(A(\varphi_1 U \varphi_2))$, for some $A(\varphi_1 U \varphi_2) \in sub(\varphi)$. This property also implies that $\vee$-nodes other than EX-nodes in such an SCC also have at most one son within the SCC, since such nodes are of the form $(s', \varphi_2 \vee (\varphi_1 \wedge AXA(\varphi_1 U \varphi_2)))$ and have two sons in the game-graph, one of which is with the subformula $\varphi_2 \notin exp(A(\varphi_1 U \varphi_2))$ and thus clearly does not belong to the SCC. Thus, every node within a non-trivial AU-SCC in C has exactly one son within the SCC and the claim is implied. $\Box$

3.2 The Annotated Counterexample is Sufficient and Minimal

Up to now we have provided an intuitive explanation for the information that is captured in an annotated counterexample. In this section we first informally describe our requirements of a counterexample. We then formalize these requirements for annotated counterexamples and show that the result of algorithm `ComputeCounter` fulfills them. Generally speaking, for a sub-model to be a counterexample, it is expected to:

1. falsify the given formula.
2. hold "enough" information to explain why the original model does not satisfy the formula.
3. be minimal, in the sense that every state and transition are needed to maintain 1 and 2.

The minimality that is expected of the counterexample is in the sense that we wish to have *precise* counterexamples, without redundancies. For example, if a finite prefix

of a path suffices to prove the refutation, we would like to see only this prefix rather than the entire (infinite) path.

The annotated counterexample is not a sub-model but a subgraph of the game-graph. Hence, the above requirements need to be adapted accordingly. Rephrasing these requirements in terms of a subgraph of the game-graph leads to the following expectations of an annotated counterexample, which correspond to the above three.

1. It contains an initial node n_0 which is colored F by χ.
2. It holds "enough" information to explain why n_0 is colored by F.
3. It is minimal, in the sense that every node and edge are needed to maintain the previous requirements.

In order to formalize the second requirement with respect to an annotated counterexample, we need the following definitions.

Definition 3.6 *Let $G = (N, E)$ be a game-graph and let A be a subgraph of G. The* partial coloring algorithm of G with respect to A *works as follows. It is given an* initial coloring function $\chi_I : N \setminus A \to \{T, F\}$ *and computes a coloring function for G. The algorithm is identical to the (original) coloring algorithm, except for the addition of the following rule:*

- *A node $n \in N \setminus A$ is colored by $\chi_I(n)$ and its color is not changed as a result of other rules.*

Any result of the partial coloring algorithm of G with respect to A is called a partial coloring function of G with respect to A, *denoted $\overline{\chi} : N \to \{T, F\}$.*

As opposed to the usual coloring algorithm that has only one possible result, referred to as the coloring function of the game-graph, the partial coloring algorithm has several possible results, depending on the initial coloring function χ_I. Each one of them is considered a partial coloring function of the game-graph w.r.t A. By definition, the usual coloring algorithm is a partial coloring algorithm of G with respect to G.

Definition 3.7 *Let G be a game-graph and let χ be the result of the coloring algorithm on G. A subgraph A of G is* independent of *G if for each $\overline{\chi}$ that is a partial coloring function of G with respect to A, and for each $n \in A$, we have that $\chi(n) = \overline{\chi}(n)$.*

Basically, a subgraph is independent of a game-graph if its coloring is *absolute* in the sense that every completion of its coloring to the full game-graph does not change the color of any node in it. In fact, one may notice that the colors of terminal nodes determine the coloring function of the full game-graph. Thus, to capture this notion,

it suffices to refer to a partial coloring algorithm that allows arbitrary coloring of the terminal nodes in $N \setminus A$, but maintains the consistency of the coloring of the rest of the nodes. However, for simplicity, we strengthen the definition and allow non-deterministic coloring of all the nodes in $N \setminus A$.

The notion of an independent subgraph captures our second expectation of an annotated counterexample, since the coloring of such a subgraph determines in a definite manner the coloring of any node within it, and in particular n_0, such that other parts of the game-graph can not affect or change it. Thus, such a subgraph holds sufficient information for explaining the color of the initial node n_0.

Having formalized the second requirement, we can now formalize the notion of an annotated counterexample. Yet, before doing so, we note that since we are dealing with formulae in negation normal form, the properties of the game-graph imply that a subgraph that fulfills the three requirements described above is in fact *entirely* colored by F (rather than just having an initial node n_0 that is colored by F). This is expressed by the following lemma.

Lemma 3.8 *Let G be a game-graph, χ its coloring function and A a subgraph of G with the following properties: (1) It contains an initial node n_0, colored F by χ, (2) It is independent of G, and (3) It is minimal. Then, for* every *$n \in A$: $\chi(n) = F$.*

Proof: Assume to the contrary that A contains at least one node that is colored T by χ. We show that removing all the nodes that are colored T from A will not affect 1 and 2. Thus, it will result in a strict subgraph of A that satisfies 1 and 2, in contradiction to the minimality of A (3).

Let $A' \subseteq A$ be the subgraph of G that results from removing all the nodes colored by T (and the corresponding edges) from A. Clearly, n_0 is not one of these nodes since $\chi(n_0) = F$ (by 1). Thus, A' contains n_0 and fulfills 1. It remains to show that it also satisfies 2, i.e. that it is independent of G.

We need to show that the partial coloring algorithm of G w.r.t A', given any initial coloring function, does not change the colors of all the nodes in A', i.e. colors them by F (by the choice of A', all the nodes in it are colored F by χ). Let $\chi'_I : N \setminus A' \to \{T, F\}$ be such an initial coloring function and let $\bar{\chi}' : N \to \{T, F\}$ be the resulting partial coloring of G w.r.t A'. We show that for every node $n' \in A'$: $\bar{\chi}'(n') = F$.

To do so, let us look at the initial coloring function $\chi''_I : N \setminus A' \to \{T, F\}$ that agrees with χ'_I on all the nodes in $N \setminus A$, but colors the nodes in $A \setminus A'$ by T. Note that $(N \setminus A) \cup (A \setminus A')$ indeed equals $N \setminus A'$. χ'_I differs from χ''_I only in (possibly) changing the colors of nodes in $A \setminus A'$ from T to F.

Note that the coloring is monotonic in the sense that changing the color of a node from T to F in the initial coloring function of $N \setminus A'$ can only cause nodes in A' to change their colors from T to F as well and not the other way around: it cannot cause their colors to change from F to T. This monotonicity holds since the game-graph is based on a CTL formula in negation normal form, thus there are no $\neg$-nodes in

it. In particular, the subgraph A' that needs to be colored consists of only terminal nodes, $\wedge$-nodes and $\vee$-nodes. The coloring of a terminal node depends on no other node and thus is not affected by the initial coloring. As for $\wedge$-nodes and $\vee$-nodes, the properties of the partial coloring algorithm assure us that an $\wedge$-node in A' is colored T iff all its sons are colored by T and an $\vee$-node in A' is colored T iff it has a son that is colored by T. Thus the monotonicity is guaranteed in such nodes as well.

As a result of the monotonicity of the coloring it suffices to show that the partial coloring function $\bar{\chi}''$ of G w.r.t A' that is based on χ_I'' colors all the nodes in A' by F. This will imply that the same holds for $\bar{\chi}'$ that results from χ_I', since χ_I' differs from χ_I'' only in possibly coloring F some nodes (in $A \setminus A'$) that are colored T by χ_I''.

It remains to show that $\bar{\chi}''$ indeed colors all the nodes in A' by F. To do so, we first use χ_I'' to construct an initial coloring function $\chi_I : N \setminus A \to \{T, F\}$ that will result in a partial coloring $\bar{\chi}$ of G w.r.t A. χ_I is defined such that for each $n \in N \setminus A$: $\chi_I(n) = \chi_I''(n)$. Note that χ_I is well defined, since $A' \subseteq A$. Since A is independent of G (2), $\bar{\chi}$ does not change the color of the nodes in A. In particular, the nodes in A' remain colored by F and the nodes in $A \setminus A'$ remain colored T. Thus, in fact $\bar{\chi}$ colors all the nodes in $N \setminus A'$ as χ_I'': for nodes in $N \setminus A$ this results from the definition of χ_I and for nodes in $A \setminus A'$ this results from the latter along with the definition of χ_I''. This implies that each execution of the partial coloring algorithm w.r.t A given χ_I (which eventually results in $\bar{\chi}$) is also a "legal" execution of it w.r.t $A' \subseteq A$ given χ_I''. Since the partial coloring algorithm is deterministic, this means that $\bar{\chi}''$, which results from the partial coloring w.r.t A' given χ_I'', colors all the nodes as $\bar{\chi}$, and in particular colors the nodes in A' by F. $\square$

We use the observation described in Lemma 3.8 and define an annotated counterexample as follows.

Definition 3.9 *Let G be a game-graph, and let χ be its coloring function, such that $\chi(n_0) = F$ for some initial node n_0. A subgraph $\tilde{C}$ of G containing n_0 is an* annotated counterexample *if it satisfies the following conditions.*

1. *For each node $n \in \tilde{C}$, $\chi(n) = F$.*
2. *$\tilde{C}$ is independent of G.*
3. *$\tilde{C}$ is minimal.*

The first two requirements in Definition 3.9 imply that $\tilde{C}$ is *sufficient* for explaining why the initial node is colored by F. First it guarantees that all the nodes in $\tilde{C}$ are colored by F. In addition, since $\tilde{C}$ is independent of G, we can conclude that regardless of the other nodes in G, all the nodes in $\tilde{C}$, and in particular the initial node n_0, will be colored by F. Therefore, it also explains why the formula is refuted by the model. The third condition shows that $\tilde{C}$ is also "necessary".

We now show that the result of algorithm `ComputeCounter`, denoted C, is indeed an annotated counterexample. The first requirement is obviously fulfilled, as

described in Lemma 3.2. The following theorems state that C satisfies the other two conditions as well.

Theorem 3.10 *C is independent of G.*

The correctness of Theorem 3.10 strongly depends on the choice of $cause(n)$ as the son of an $\wedge$-node in the algorithm `ComputeCounter`. Returning to the example presented in Figure 3.1, the subgraph in Figure 3.1(c) which was already informally claimed not to provide a "good" annotated counterexample, can now be formally shown to be *not* independent of G. For example, an initial coloring function χ_I that colors the node $(s_1, A(pVq))$ by T, would result in a partial coloring function of G w.r.t the subgraph from Figure 3.1(c), where the nodes $(s, A(pVq)$, $(s, q \wedge (p \vee AXA(pVq)))$, $(s, p \vee AXA(pVq))$ and $(s, AXA(pVq))$ from this subgraph, are colored by T instead of F.

For the proof of Theorem 3.10, we need the following technical Lemma.

Lemma 3.11 *Let n be a node in C that was colored due to a witness during the partial coloring of G with respect to C, given an initial coloring function χ_I. Suppose that all the nodes from C that were colored prior to n by the partial coloring algorithm were colored by F. Then n lies on a cycle in C.*

Proof: Suppose n was colored due to a witness by the partial coloring algorithm of G. Let us look at the status of the game-graph at the phase of the partial coloring algorithm, where n was colored by a witness. Obviously, n has a son that is not yet colored at that time (otherwise n would be colored too, based on its sons). This son must be within the same set Q_i (all the sons outside Q_i are in "smaller" sets Q_j and are thus already colored). Let us show that at least one such uncolored son is in C.

- If n is an $\vee$-node, then all of its sons are in C, and in particular the uncolored one, which concludes this case.
- If n is an $\wedge$-node, then it has exactly one son n' in C. If this son n' is not one of the uncolored ones, then it is already colored at this time, thus by our assumption it is already colored by F, which would cause n to be already colored by F too (based on this son rather than on a witness) in contradiction.

In any case, we get that each of these types of nodes has a son within its Q_i that is also in C and is not yet colored by the partial coloring algorithm. Such a son will be colored due to a witness along with n. The same arguments apply to this son and to its son, etc. Since there is a finite number of nodes in Q_i, we get a cycle of such nodes, which are all in C. $\Box$

We now return to the proof of Theorem 3.10.

Proof of Theorem 3.10: We need to show that in any partial coloring function of G with respect to C, all the nodes of C are colored as they were originally colored by χ, i.e. by F. Thus, for each node $n = (s, \varphi') \in C$ we prove that n is colored F by the partial coloring algorithm with respect to C, regardless of the initial coloring. The proof is by induction on the execution of the partial coloring algorithm.

Base case:

1. $n \in N \setminus C$: we have nothing to prove.
2. $n \in C$ is colored as a terminal node (leaf). Since $n \in C$, we have that it was originally colored F by χ, which means φ' is either $l \notin L(s)$ or false. Thus n is again colored F by the partial coloring algorithm.

Induction step:

- $n \in C$ is colored due to its sons by the partial coloring algorithm. We distinguish between two possibilities.

  1. n is an $\vee$-node.
     Suppose n is colored by T by the partial algorithm. This means that it has at least one son n' that is already colored by T. However, since $n \in C$ is an $\vee$-node, then by the construction of the annotated counterexample C, all of its sons are in C, and in particular $n' \in C$. Thus the induction hypothesis, applied to n' which was already colored by our assumption, assures us that n' is colored by F. This contradicts our assumption, i.e. n must be colored by F.
  2. n is an $\wedge$-nodes.
     Suppose n is colored by T. This means that all of its sons are already colored by T. However, since $n \in C$, then by the construction of the annotated counterexample, n has exactly one son n' in C. Thus, the induction hypothesis, applied to n' which was already colored by our assumption, assures us that n' is colored by F. This contradicts our assumption that all the sons of n, and in particular n', are colored by T, i.e. n must be colored by F.

- $n \in C$ is colored due to a witness. Since it is colored due to a witness, we have that it could not be colored based on its sons only. If n is colored due to the existence of an AU or EU witness, then by the description of the algorithm, it is colored by F, as required.

  It remains to show that the witness cannot be AV or EV, which would have caused n to be colored by T. Let us rule out these possibilities by assuming the contrary. Suppose n is colored due to a witness of the form AV or EV by the partial coloring algorithm. The induction hypothesis provides the information

that is needed in order to use Lemma 3.11. According to Lemma 3.11, we get that n lies on a cycle of nodes from C, which forms a non-trivial SCC. However, by Corollary 3.4, non-trivial SCCs in C are either AU-SCCs or EU-SCCs, which contradicts the assumption that n is colored due to an AV or EV witness (By Lemma 2.6, such a witness cannot exist in an AU or EU SCC). Thus the witness that caused the coloring of n cannot be one of the above.

□

The following Theorem refers to the minimality of C.

Theorem 3.12 *C is* minimal *in the sense that removing a node or an edge will result in a subgraph that is not independent of G.*

Proof: It suffices to show that any node and edge that will be removed from C will result in a subgraph C' that the partial coloring of G with respect to it may change its coloring. This property is guaranteed because of the following. If the son of an $\wedge$-node n (or the edge that connects them) is removed, then there is no longer a son for this node in C', thus there exists an initial coloring function (input for the partial coloring algorithm) that colors all the sons of n by T, which will cause n to become colored T by the partial coloring algorithm. If a son of an $\vee$-node n is removed, then this son can be colored by T by the initial coloring function (input of the partial coloring algorithm), thus its parent will also be colored T by the partial coloring algorithm. □

3.3 Practical Considerations

Since the computed annotated counterexample C may be big and difficult to understand, several simplifications may be suggested in order to make it smaller, and thus easier to navigate and to comprehend.

Zoom In - Zoom Out

Since each non-trivial SCC in C is "attached" to a single AU or EU formula, as indicated by Corollary 3.4, the annotated counterexample can be "zoomed-out" into a DAG. This can be done by constructing the (maximal strongly connected) components graph of the annotated counterexample, where each non-trivial MSCC is replaced by a single node, annotated with its witness. This way, we allow a "global" view on the annotated counterexample, along with the possibility to "zoom in" into each MSCC and view its inner structure. This allows a user to interact with the system and navigate through different parts of the annotated counterexample.

Reductions of the Annotated Counterexample

The complexity of the annotated counterexample results, at least partly, from the node annotations and the auxiliary edges, that add information about the formula. This auxiliary information is valuable in the sense that it helps in understanding the counterexample. However, the annotated counterexample can be "reduced" into more compact structures by hiding (some of) the information that results from the formula. Auxiliary edges may be collapsed by merging nodes that were originally separated by them, resulting in a subgraph of the unwinded model. Node annotations can either be removed or partially remembered. These simplifications reflect the trade-off between the size of the counterexample and the additional information originating from the formula.

Presenting *All* the Possible Counterexamples

The algorithm `ComputeCounter` produces a *single* annotated counterexample. Yet, the colored game-graph holds the *full* information about the refutation of the formula. This information is described by the reachable subgraph that is colored by F. Thus, we can identify *all* the possible annotated counterexamples. It is possible to (interactively) give them all using a variation of the algorithm `ComputeCounter`.

Chapter 4

Game-Based Model Checking for Abstract Models

In this chapter, we extend the discussion to abstract models. We suggest a generalization of the game-based model checking algorithm for evaluating a CTL formula φ over a KMTS M (that represents an abstract model) according to the 3-valued semantics. In Appendix A we describe the abstract *2-valued semantics* of CTL, which is also suitable for KMTSs. We present an abstract model checking algorithm based on the 2-valued semantics and discuss solving the 3-valued problem by reducing it to two instances of the 2-valued problem, as suggested in [22]. We also discuss the advantages of the direct solution, described in this chapter.

We start with the description of the 3-valued game. The main difference arises from the fact that KMTSs have two types of transitions. Since the transitions of the model are considered only in configurations with subformulae of the form $AX\varphi_1$ or $EX\varphi_1$, these are the only cases where the rules of the play need to be changed. Intuitively, in order to be able to both prove and refute each subformula, the game needs to allow the players to use both may and must transitions in such configurations. The reason is that for example, truth of a formula $AX\varphi_1$ should be checked upon may-transitions, but its falseness should be checked upon must-transitions. Therefore, the new moves of the game are adapted as follows.

The new moves of the game:

2. if $C_i = (s, AX\varphi)$, then $\forall$belard chooses a transition $s \xrightarrow{\text{must}} s'$ (for refutation) or $s \xrightarrow{\text{may}} s'$ (for satisfaction), and $C_{i+1} = (s', \varphi)$.

3. if $C_i = (s, EX\varphi)$, then $\exists$loise chooses a transition $s \xrightarrow{\text{must}} s'$ (for satisfaction) or $s \xrightarrow{\text{may}} s'$ (for refutation), and $C_{i+1} = (s', \varphi)$.

That is, each player can use both may and must transitions. Intuitively, the players use must-transitions in order to win, while they use may-transitions in order to prevent the other player from winning. As a result it is possible that none of

the players wins the play, i.e. the play ends with a *tie*. As before, a *maximal* play, defined in Chapter 2, is infinite if and only if exactly one *witness*, which is either an AU,EU,AV or EV-formula, appears in it infinitely often. However, the winning rules become more complicated. A player can only win the play if he or she are "consistent" in their moves: always makes moves that are designed for satisfaction (if the player is $\exists$loise), or always makes moves that are designed for refutation (if it is $\forall$belard). These moves are all based on must transitions. The other player, on the other hand, possibly uses both types of transitions.

Definition 4.1 *1. A play is called* true-consistent *if in each configuration of the form $C_i = (s, EX\varphi)$, $\exists$loise chooses a move based on $\stackrel{must}{\longrightarrow}$ transitions.*

2. A play is called false-consistent *if in each configuration of the form $C_i = (s, AX\varphi)$, $\forall$belard chooses a move based on $\stackrel{must}{\longrightarrow}$ transitions.*

The new winning criteria:

- $\forall$belard wins the play iff the play is false-consistent and in addition one of the following holds:
  1. the play is finite and ends in a terminal configuration of the form $C_i = (s, \text{false})$, or $C_i = (s, l)$, where $\neg l \in L(s)$.
  2. the play is infinite and the witness is of the form AU or EU.
- $\exists$loise wins the play iff the play is true-consistent and in addition one of the following holds:
  1. the play is finite and ends in a terminal configuration of the form $C_i = (s, \text{true})$, or $C_i = (s, l)$, where $l \in L(s)$.
  2. the play is infinite and the witness is of the form AV or EV.
- Otherwise, the play ends with a tie.

We now have the following correspondence between the game and the truth value of a formula in a certain state under the 3-valued semantics.

Theorem 4.2 *Let M be a KMTS and φ a CTL formula. Then, for each $s \in S$:*

1. $[(M,s) \stackrel{3}{\models} \varphi] = tt$ *iff* *$\exists$loise has a winning strategy starting at (s, φ).*
2. $[(M,s) \stackrel{3}{\models} \varphi] = ff$ *iff* *$\forall$belard has a winning strategy starting at (s, φ).*
3. $[(M,s) \stackrel{3}{\models} \varphi] = \perp$ *iff* *none of the players has a winning strategy from (s, φ).*

Proof: The proof is by induction on the structure of CTL formulae. It suffices to prove the implication from the truth value to the existence of a winning strategy.

Base case: φ = true, false or l, where $l \in Lit$. Thus, $C_0 = (s, \varphi)$ is a terminal configuration. By the winning criteria, if $[s \models^3 \varphi] = \mathrm{tt}$, then $\exists$loise wins every play, right at the beginning. If $[s \models^3 \varphi] = \mathrm{ff}$, then $\forall$belard wins every play, right at the beginning. Otherwise, every play ends with a tie, and therefore none of them has a winning strategy.

Induction step:

- $\varphi = \varphi_1 \vee \varphi_2$:

  1. If $[s \models^3 \varphi] = \mathrm{tt}$, then by the definition of the 3-valued semantics, there exists $j \in \{1, 2\}$ such that $[s \models^3 \varphi_j] = \mathrm{tt}$. By the induction hypothesis, $\exists$loise has a winning strategy for every play that starts from (s, φ_j). Thus, her winning strategy, starting from (s, φ), consists of choosing the next move, which is in her responsibility, to be (s, φ_j) and from then on, using the guaranteed winning strategy for (s, φ_j).

  2. If $[s \models^3 \varphi] = \mathrm{ff}$, then by the semantics definition, for each $j \in \{1, 2\}$ we have that $[s \models^3 \varphi_j] = \mathrm{ff}$. By the induction hypothesis, $\forall$belard has a winning strategy for every play that starts from either one of the configurations (s, φ_j). The union of these winning strategies is his winning strategy for the game starting from (s, φ). This is indeed a winning strategy, because no matter which of the two possible configurations is chosen by $\exists$loise as her move, $\forall$belard has a winning strategy from this point and on.

  3. If $[s \models^3 \varphi] = \perp$, then by the definition, at least for one of $j \in \{1, 2\}$ we have that $[s \models^3 \varphi_j] = \perp$, and for the other one, $k \in \{1, 2\}$, $k \neq j$, we have that the value of $[s \models^3 \varphi_k]$ is $\perp$ or ff. Thus, $\forall$belard has no winning strategy because $\exists$loise, that makes the move in the initial configuration, can always choose to proceed to (s, φ_j), for which $\forall$belard has no winning strategy, by the induction hypothesis. In addition, $\exists$loise has no winning strategy, because no matter which move she makes, she reaches a configuration for which by the induction hypothesis she does not have a winning strategy.

- $\varphi = \varphi_1 \wedge \varphi_2$: symmetric, since the definition of 3-valued semantics is opposite and so are the roles of the players in a configuration with such a formula.

- $\varphi = EX\varphi_1$:

  1. If $[s \models^3 \varphi] = \mathrm{tt}$, then by the semantics definition, there exists a transition $s \xrightarrow{\text{must}} s'$, such that $[s' \models^3 \varphi_1] = \mathrm{tt}$. By the induction hypothesis, $\exists$loise has a winning strategy for every play that starts from (s', φ_1). Thus, her

winning strategy, starting from (s, φ), consists of choosing the next move, which is in her responsibility, to be (s', φ_1), which allows her to maintain the play true-consistent, and from then on, using the guaranteed winning strategy for (s, φ_j).

2. If $[s \models^3 \varphi] = \text{ff}$, then by the semantics definition, for each transition $s \xrightarrow{\text{may}} s'$, we have that $[s' \models^3 \varphi_1] = \text{ff}$. Recall that $\xrightarrow{\text{must}} \subseteq \xrightarrow{\text{may}}$, thus this applies to $\xrightarrow{\text{must}}$ transitions as well. By the induction hypothesis, $\forall$belard has a winning strategy for every play that starts from either one of these configurations (s', φ_1). The union of these winning strategies is his winning strategy for the game starting from (s, φ). This is indeed a winning strategy, because these are the only possibilities that $\exists$loise has for the first move, and no matter which of the possible configurations is chosen by her, $\forall$belard has a winning strategy from this point and on.

3. If $[s \models^3 \varphi] = \bot$, then by the definition, there exists at least one transition $s \xrightarrow{\text{may}} s'$ such that the value of $[s' \models^3 \varphi_1]$ is $\bot$ or tt, and for all the transitions $s \xrightarrow{\text{must}} s''$, we have that the value of $[s'' \models^3 \varphi_1]$ is $\bot$ or ff. Thus, $\forall$belard has no winning strategy because $\exists$loise, that makes the move in the initial configuration, can always choose to proceed to (s', φ_1), for which $\forall$belard has no winning strategy, by the induction hypothesis. In addition, $\exists$loise has no winning strategy, because for her to win, the play must be true-consistent, and thus she has to choose as her first move one of the configurations (s'', φ_1), which are based on must-transitions. However, no matter which of them she chooses, she reaches a configuration for which by the induction hypothesis she does not have a winning strategy.

- $\varphi = AX\varphi_1$: symmetric, since the definition is opposite and so are the roles of the players.

- $\varphi = A(\varphi_1 U \varphi_2)$:

  First note that (the prefix of) each play that starts from the configuration (s, φ) is of the *periodic* form $(s, A(\varphi_1 U \varphi_2)) \to (s, \varphi_2 \vee (\varphi_1 \wedge AXA(\varphi_1 U \varphi_2))) \to_\exists (s, \varphi_1 \wedge AXA(\varphi_1 U \varphi_2)) \to_\forall (s, AXA(\varphi_1 U \varphi_2)) \to_\forall (s_1, A(\varphi_1 U \varphi_2)) \to \ldots$, based on some (must or may) path $s, s_1, \ldots$ from s. This form continues as long as none of the players chooses as a next move (s_i, φ_2) ($\exists$loise) or (s_i, φ_1) ($\forall$belard) from the configurations $(s_i, \varphi_2 \vee (\varphi_1 \wedge AXA(\varphi_1 U \varphi_2)))$ or $(s_i, \varphi_1 \wedge AXA(\varphi_1 U \varphi_2))$ respectively.

  If a player chooses as a next move (s_i, φ_1) or (s_i, φ_2), we say that he or she *interrupts* the periodic form of the play in index i. Otherwise, we say that he or she *maintains* the periodic form of the play in index i.

  In addition note that in fact $\forall$belard is responsible for choosing the path that the play is based on, because in configurations of the form $(s_i, AXA(\varphi_1 U \varphi_2))$ where the move is based on a transition of the model, $\forall$belard is the player that makes the move. If $\forall$belard bases his moves on a path $\pi = s_0, s_1, \ldots$ and

maintains the periodic form of the play in his move in index i, then this means that he continues from $(s_i, \varphi_1 \wedge AXA(\varphi_1 U \varphi_2))$ to $(s_i, AXA(\varphi_1 U \varphi_2))$ and to $(s_{i+1}, A(\varphi_1 U \varphi_2))$. In this case we say that $\forall$belard *proceeds along* π in index i.

1. If $[s \models^3 \varphi] = \text{tt}$, then by the semantics definition, for each may-path $\pi = s_0, s_1, \ldots$ from s, we have that $[\pi \models^3 \varphi_1 U \varphi_2] = \text{tt}$. This means that for each such path there exists $0 \leq k \leq |\pi|$ such that $[s_k \models^3 \varphi_2] = \text{tt}$ and for all $j < k$: $[s_j \models^3 \varphi_1] = \text{tt}$. Thus, by the induction hypothesis, $\exists$loise has a winning strategy for each game that starts from either one of the corresponding configurations (s_k, φ_2) and (s_j, φ_1). The winning strategy of $\exists$loise for a game that starts from the configuration (s, φ) consists of all these winning strategies, along with the rules that tell her to interrupt the periodic form of the play and proceed to (s_i, φ_2) if $i = k$ for some k as described above, and to maintain the periodic form if $i < k$. This is a winning strategy, because as long as $i < k$ for some k as described above, if $\forall$belard chooses (s_i, φ_1), then since $[s_i \models^3 \varphi_1] = \text{tt}$, by the induction hypothesis $\exists$loise has a winning strategy. Otherwise, $\forall$belard maintains the periodic form of the play in such i's, and so does $\exists$loise (as described by her strategy). When $i = k$ is reached, $\exists$loise chooses the configuration (s_k, φ_2), for which she has a winning strategy by the induction hypothesis, since $[s_k \models^3 \varphi_2] = \text{tt}$.
2. If $[s \models^3 \varphi] = \text{ff}$, then by its definition, there exists a must-path $\pi = s_0, s_1, \ldots$ from s, such that $[\pi \models^3 \varphi_1 U \varphi_2] = \text{ff}$. This means that $\big(\forall\ 0 \leq k \leq |\pi| : [(\forall j < k : [s_j \models^3 \varphi_1] \neq \text{ff}) \Rightarrow ([s_k \models^3 \varphi_2] = \text{ff})]\big) \wedge \big((\forall\ 0 \leq k \leq |\pi| : [s_k \models^3 \varphi_1] \neq \text{ff}) \Rightarrow |\pi| = \infty\big)$. If there exists $0 \leq i \leq |\pi|$ for which $[s_i \models^3 \varphi_1] = \text{ff}$, then let k be the *smallest* index for which $[s_k \models^3 \varphi_1] = \text{ff}$. Otherwise, we set $k = \infty$. Note that in this case $|\pi| = \infty$ as well. Either way, by the minimality of k, we know that for every $i \leq k$ the formula $\forall j < i : [s_j \models^3 \varphi_1] \neq \text{ff}$ holds, which implies that $[s_i \models^3 \varphi_2] = \text{ff}$. Therefore, for every $i \leq k$ we have that $[s_i \models^3 \varphi_2] = \text{ff}$.
   By the induction hypothesis, $\forall$belard has a winning strategy from every configuration (s_i, φ_2) for which $[s_i \models^3 \varphi_2] = \text{ff}$ and from every configuration (s_i, φ_1) for which $[s_i \models^3 \varphi_1] = \text{ff}$. Thus, his winning strategy for a game that starts from the configuration (s, φ) includes these winning strategies, as well as rules that tell him to interrupt the periodic form of the play and proceed to (s_i, φ_1) if $i = k$, and to proceed along π otherwise. This is a winning strategy, because as long as $i \leq k$, if $\exists$loise chooses (s_i, φ_2), then by the induction hypothesis $\forall$belard has a winning strategy (which is part of his winning strategy from (s, φ)), since $[s_i \models^3 \varphi_2] = \text{ff}$. Otherwise, $\exists$loise maintains the periodic form of the play in such i's. As for $\forall$belard, as long as $i < k$, he proceeds along the must-path π, maintaining the play false-consistent. Now, there are two possibilities:

(a) If k is finite, then when $i = k$ is reached, by the described strategy $\forall$belard chooses the configuration (s_k, φ_1) as his next move. Since by the choice of k $[s_k \models^3 \varphi_1] =$ ff, then by the induction hypothesis $\forall$belard has a winning strategy from this configuration.

(b) Otherwise, we have that the must-path π itself is infinite and for every $i \geq 0$ $\forall$belard *proceeds along* π, such that the play is infinite and false-consistent with the AU-witness. Thus, $\forall$belard wins as well.

3. If $[s \models^3 \varphi] = \bot$, then by the semantics definition, there exists a may-path π from s, for which the value of $[\pi \models^3 \varphi_1 U \varphi_2]$ is $\bot$ or ff, and in addition for each must-path π' the value of $[\pi' \models^3 \varphi_1 U \varphi_2]$ is $\bot$ or tt.

   The existence of π shows that $\exists$loise does not have a winning strategy. Let $\pi = s_0, s_1 \ldots$, then by the definition of the semantics $\neg\exists\ 0 \leq k \leq |\pi| : [([s_k \models^3 \varphi_2] = \text{tt}) \wedge (\forall j < k : [s_j \models^3 \varphi_1] = \text{tt})]$. That is, $\forall\ 0 \leq k \leq |\pi| : [([s_k \models^3 \varphi_2] \neq \text{tt}) \vee (\exists j < k : [s_j \models^3 \varphi_1] \neq \text{tt})]$. In addition, since π is a *may* path, then it is infinite ($|\pi| = \infty$). Now, in order to prevent $\exists$loise from winning, $\forall$belard can base the play on π. We consider two possibilities.

   (a) There exists ($|\pi| \geq$) $i \geq 0$ such that $[s_i \models^3 \varphi_1] \neq \text{tt}$. In this case we set k to be the smallest such index, thus for each $i \leq k$ we have that $\neg\exists j < i : ([s_j \models^3 \varphi_1] \neq \text{tt})$, which means that for each $i \leq k : [s_i \models^3 \varphi_2] \neq \text{tt}$. In this case, $\forall$belard can always choose to proceed along π as long as $i < k$, and when $i = k$ he can choose the configuration (s_k, φ_1). If $\exists$loise interrupts the periodic form of the play before (or when) $i = k$, then it is to a configuration (s_i, φ_2), for which by the induction hypothesis she does not have a winning strategy (since for each $i \leq k : [s_i \models^3 \varphi_2] \neq \text{tt}$). Otherwise, when (s_k, φ_1) is reached, she does not have a winning strategy (by the induction hypothesis, since $[s_k \models^3 \varphi_1] \neq \text{tt}$).

   (b) Otherwise, for every $i \geq 0 : [s_i \models^3 \varphi_1] = \text{tt}$. In this case we get that for every $i \geq 0 : [\neg\exists j < i : ([s_j \models^3 \varphi_1] \neq \text{tt})]$. Thus, for each $i \geq 0$, $[s_i \models^3 \varphi_2] \neq \text{tt}$. Hence, if $\exists$loise chooses the configuration (s_i, φ_2) as her next move at any point, then she reaches a configuration for which she does not have a winning strategy (by the induction hypothesis). If $\exists$loise never chooses (s_i, φ_2), then the play is infinite (since π is infinite) with the witness $A(\varphi_1 U \varphi_2)$, thus $\exists$loise can not win.

   The property of each must-path shows that $\forall$belard does not have a winning strategy. Let $\pi' = s'_0, s'_1 \ldots$ denote such a path. Then the value of $[\pi' \models^3 \varphi_1 U \varphi_2]$ is either $\bot$ or tt, which means that $\neg\big(\forall\ 0 \leq k \leq |\pi'| : [(\forall j < k : [s'_j \models^3 \varphi_1] \neq \text{ff}) \Rightarrow ([s'_k \models^3 \varphi_2] = \text{ff})]\big) \vee \neg\big((\forall\ 0 \leq k \leq |\pi'| : [s'_k \models^3 \varphi_1] \neq \text{ff}) \Rightarrow |\pi'| = \infty\big)$ holds. In other words, $\big(\exists\ 0 \leq k \leq |\pi'| : [(\forall j < k : [s'_j \models^3 \varphi_1] \neq \text{ff}) \wedge ([s'_k \models^3 \varphi_2] \neq \text{ff})]\big) \vee \big((\forall\ 0 \leq k \leq |\pi'| : [s'_k \models^3 \varphi_1] \neq \text{ff}) \wedge |\pi'| \neq \infty\big)$

holds. This means that for every such must path π' there exists $k \geq 0$ such that either $(k \leq |\pi'|) \wedge (\forall j < k : [s'_j \stackrel{3}{\models} \varphi_1] \neq \text{ff}) \wedge ([s'_k \stackrel{3}{\models} \varphi_2] \neq \text{ff})$ (if the first disjunct holds), or $(k = |\pi'|) \wedge (\forall j \leq k : [s'_j \stackrel{3}{\models} \varphi_1] \neq \text{ff})$ (if the second disjunct holds). In order to win, $\forall$belard needs to proceed along must-sons of AX-nodes (for the play to be false-consistent), thus in particular as long as the play has the periodic form, $\forall$belard needs to base it on a must-path $\pi' = s'_0, s'_1 \ldots$. Clearly, if $\forall$belard interrupts the periodic form of the play before k (of the relevant must path) is reached, then it is to a configuration (s'_j, φ_1) for which by the induction hypothesis he does not have a winning strategy (since for $j < k$ we know that $s'_j \stackrel{3}{\models} \varphi_1] \neq \text{ff}$). Otherwise, $\forall$belard maintains the periodic form of the play for every $j < k$, and in order to prevent him from winning, $\exists$loise will do the same. Now, when k is reached we have two possibilities.

(a) If $[s'_k \stackrel{3}{\models} \varphi_2] \neq \text{ff}$, then $\exists$loise can interrupt the periodic form of the play at this point and proceed to (s'_k, φ_2), for which $\forall$belard does not have a winning strategy by the induction hypothesis.

(b) Otherwise, we have that $|\pi'| = k$ and we also know that $[s'_k \stackrel{3}{\models} \varphi_1] \neq \text{ff}$. In this case $\exists$loise will maintain the periodic form of the play at this point as well. Now, if $\forall$belard at his turn chooses to interrupt the periodic form of the play, then he will reach the configuration (s'_k, φ_1) for which he does not have a winning strategy (by the induction hypothesis). Otherwise, he will proceed to the configuration $(s'_k, AXA(\varphi_1 U \varphi_2))$ and at this point he will not be able to base his move on a must transition, because $|\pi'| = k$ (and π is maximal by definition, which means that there is no must transition that exits the state s'_k). Therefore, $\forall$belard will be forced to proceed based on a *may* transition that is not a must transition. As a result, the play will not be false-consistent and $\forall$belard will not win.

- $\varphi = E(\varphi_1 U \varphi_2)$, $A(\varphi_1 V \varphi_2)$ or $E(\varphi_1 V \varphi_2)$: similar.

□

Theorem 4.2 refers to the existence (or non-existence) of winning strategies for the players. One may also be interested in the existence of *memoryless* winning strategies. The interested reader is referred to appendix B, where we define the notion of a *memoryless* strategy and it is shown that Theorem 4.2 can be rephrased to refer to *memoryless* winning strategies.

In order to use the correspondence described in Theorem 4.2 for model checking, we generalize the game-based model checking algorithm.

4.1 Game-Graph Construction and its Properties

The construction of the (3-valued) game-graph, denoted $G_{M\times\varphi}$, is defined as for the "concrete" game (as described in Chapter 2). The nodes of the game-graph, denoted N, can again be classified as $\wedge$-nodes, $\vee$-nodes, AX-nodes and EX-nodes. Similarly, the edges can be classified as *progress* edges, that originate in AX or EX nodes, or *auxiliary* edges. But now, we distinguish between two types of progress edges, two types of sons and two types of SCCs.

- Edges that are based on must-transitions are referred to as *must-edges*. Edges that are based on may-transitions are referred to as *may-edges*.
- A node n' is a *may-son* of the node n if there exists a may-edge (n, n'). n' is a *must-son* of n if there exists a must-edge (n, n').
- An SCC in the game-graph is a *may-SCC* if all its progress edges are may-edges. It is a *must-SCC* if all its progress edges are must-edges.

4.2 Coloring Algorithm

The coloring algorithm of the 3-valued game-graph needs to be adapted as well. First, a new color, denoted ?, is introduced for configurations in which none of the players has a winning strategy.

Second, the partition to Q_i's that is based on MSCCs is affected because there are two types of SCCs (and MSCCs) in $G_{M\times\varphi}$. However, $\xrightarrow{\text{must}} \subseteq \xrightarrow{\text{may}}$, thus each must-edge is also a may-edge and every must-SCC is also a may-SCC. As a result, the graph can be partitioned to may-MSCCs (based on the may-edges). Note that Lemma 2.6 holds for may-SCCs in the 3-valued game-graph as well. Thus, the notion of a *witness* in an SCC is also valid.

As for the coloring itself, similarly to the concrete case, the 3-valued coloring algorithm processes the game-graph bottom-up and colors nodes by T, F or ? based on the colors of their sons, according to the type of the node ($\wedge$ versus $\vee$, AX versus EX) and the (3-valued) semantics of $\wedge$, $\vee$, AX and EX. Here too, if at any point the coloring cannot proceed, then the existence of a non-trivial SCC and a corresponding witness is implied. Yet, in this case further analysis is needed before being able to color the remaining nodes. Intuitively, if the witness is of the form AU, then uncolored loops can only be used to prove its refutation (F color). To do so, "real" loops are needed. Thus for such a witness, we need to have an uncolored non-trivial *must-SCC* in order to color it by a definite color (F). On the other hand, for an AV-witness, loops can contribute to satisfaction, and satisfaction of universal properties should be examined upon may-transitions. Thus for such a witness, *may-edges* are sufficient to color the loop by T. Similarly, if the witness is of the form EU, we need to have a may-SCC, whereas for an EV witness, a must-SCC is used. Thus, unlike the concrete

case where all the remaining uncolored nodes were colored automatically according to the witness, in this case we first apply a special method whose purpose is to ensure that the remaining uncolored nodes form non-trivial SCCs with the type that is required for a definite color. During this process, some of the remaining uncolored nodes, which do not fill these criteria, are colored by ?. Only after this phase, the remaining nodes are colored by the suitable definite color.

The (3-valued) coloring algorithm

Partition and Order: The game-graph is partitioned into its may-MSCCs. The resulting sets are denoted Q_i's. This partition induces a partial order such that transitions go out of a set only to itself or to a "smaller" set. The partial order is extended to a total order $\leq$ arbitrarily.

Coloring: As before, the coloring algorithm processes the Q_i's according to $\leq$, bottom-up. Let Q_i be the smallest set w.r.t $\leq$ that is not yet fully colored. The nodes of Q_i are colored in two phases, as follows.

1. *Sons-coloring phase:*

   Apply the following rules to all the nodes in Q_i until none of them is applicable.

   - A terminal node is colored by T if $\exists$loise wins in it, by F if $\forall$belard wins in it, and by ? otherwise.
   - An AX-node is colored by:
     - T if all its may-sons are colored by T.
     - F if it has a must-son that is colored by F.
     - ? if all its must sons are colored by T or ? and it has a may-son that is colored by F or ?.
   - An EX-node is colored by:
     - T if it has a must-son that is colored by T.
     - F if all its may-sons are colored by F.
     - ? if it has a may-son that is colored by T or ? and all its must-sons are colored by F or ?.
   - An $\wedge$-node, other than AX-node, is colored by:
     - T if both its sons are colored by T.
     - F if it has a son that is colored by F.
     - ? if it has a son that is colored ? and the other one is colored ? or T.
   - An $\vee$-node, other than EX-node, is colored by:
     - T if it has a son that is colored by T.
     - F if both its sons are colored by F.
     - ? if it has a son that is colored ? and the other one is colored ? or F.

Note that auxiliary edges can be considered both may and must edges. By doing so, it is possible to join the description of the coloring of AX-nodes and other $\wedge$-nodes, as well as the description of EX-nodes and other $\vee$-nodes. We do not do that for clarity.

2. *Witness-coloring phase:*

   If after the propagation of the rules of phase 1 there are still nodes in Q_i that remain uncolored, then Q_i must be a non-trivial may-MSCC that has exactly one witness (by Lemma 2.6 which holds here as well). The uncolored nodes in Q_i are colored according to the witness in two phases, as follows.

   - The witness is of the form $A(\varphi_1 U \varphi_2)$ or $E(\varphi_1 U \varphi_2)$:
     (a) Repeatedly color by ? each node in Q_i that satisfies one of the following conditions, until there is no change (i.e. none of the conditions holds for any node in Q_i).
     - An AX-node that all its must sons are colored by T or ?.
     - An $\wedge$-node that both its sons are colored by T or ?.
     - An EX-node that has a may son that is colored by T or ?.
     - An $\vee$-node that has a son that is colored by T or ?. (or equivalently: An $\vee$-node that has a son that is colored by ?, since the T option is impossible).

     In fact, each node for which the F option is no longer possible according to the rules of the *sons-coloring* phase is colored by ?.
     (b) Color the remaining nodes in Q_i by F.
   - The witness is of the form $A(\varphi_1 V \varphi_2)$ or $E(\varphi_1 V \varphi_2)$:
     (a) Repeatedly color by ? each node in Q_i that satisfies one of the following conditions, until there is no change (i.e. none of the conditions holds for any node in Q_i).
     - An AX-node that has a may son that is colored by F or ?.
     - An $\wedge$-node that has a son that is colored by F or ?. (or equivalently: An $\wedge$-node that has a son that is colored by ?, since the F option is impossible).
     - An EX-node that all its must sons are colored by F or ?.
     - An $\vee$-node that both its sons are colored by F or ?.

     In fact, each node for which the T option is no longer possible according to the rules of the *sons-coloring* phase is colored by ?.
     (b) Color the remaining nodes in Q_i by T.

The result of the coloring algorithm is a *3-valued coloring function* $\chi : N \to \{T, F, ?\}$.

Note that coloring by ? is done carefully. One may suggest to color a node by ? in any case where it is not colored and the coloring can not proceed. However, since we

would like to follow the 3-valued semantics, we need to color such a node by ? only if it is *not possible* that it should be colored by T or F. This is not implied by the node being uncolored. Therefore, a node is colored by ? only if there is *evidence* that it cannot be colored otherwise. In any other case, another method is used to determine its color.

We now discuss the correctness of the coloring algorithm. To do so, we first prove the following important Lemma.

Lemma 4.3 *Let n be a node that is uncolored at the beginning of phase 2 in its set Q_i. Then n lies on a non-trivial SCC that is a subgraph of Q_i and all nodes of the SCC are uncolored at the beginning of phase 2.*

We conclude that if a set Q_i has uncolored nodes at the beginning of phase 2 then Q_i is a non-trivial *may-MSCC.*

Proof: Consider an uncolored node in Q_i, denoted n. n has outgoing edges only to nodes in smaller sets Q_j's, which are already colored, or to nodes in the same set Q_i (by the choice of the order $\leq$). Since n is uncolored, we have that it has an outgoing edge to an uncolored node n', otherwise it could be colored in phase 1. This node n' can only be in the same set Q_i (the others are already colored). Thus, each uncolored node in Q_i has a son within Q_i that is not colored. Since Q_i is finite, this results in a non-trivial SCC, whose nodes are all within Q_i.

We now conclude that in this case Q_i is a non-trivial may-MSCC. By the choice of the partition of $G_{M\times\varphi}$, Q_i is obviously a may-MSCC. Since we have seen that it contains a non-trivial SCC, we can also conclude that Q_i itself is a non-trivial may-MSCC. □

Theorem 4.4 *All the nodes in the game-graph get colored by the 3-valued coloring algorithm.*

Proof: It suffices to prove that whenever phase 2 of the coloring is reached (with the existence of uncolored nodes), the set Q_i that is handled by the algorithm indeed has exactly one witness. This results from Lemma 4.3 that guarantees that Q_i is a non-trivial MSCC, as well as Lemma 2.6, that guarantees that a non-trivial may-MSCC has exactly one witness. Thus, all the remaining uncolored nodes in Q_i are colored according to this witness. As a conclusion, we get that no node is left uncolored. □

Theorem 4.5 *Let $G_{M\times\varphi}$ be a 3-valued game-graph and let n be a node in the game-graph, then:*

1. $\chi(n) = T$ *iff* $\exists$*loise has a winning strategy starting at n.*
2. $\chi(n) = F$ *iff* $\forall$*belard has a winning strategy starting at n.*

3. $\chi(n) = ?$ *iff none of the players has a winning strategy starting at* n.

The correctness of Theorem 4.5 strongly depends on the observation described by the following Lemma.

Lemma 4.6 *Let Q_i be the set that is handled by the coloring algorithm at its ith iteration, and let n be a node in Q_i that is uncolored at the beginning of phase 2b. Then*

1. *If Q_i has an AU or EV witness, then n lies on a non-trivial* must-SCC *that is a subgraph of Q_i and all nodes of the must-SCC are uncolored at the beginning of phase 2b.*

2. *If Q_i has an EU or AV witness, then n lies on a non-trivial* may-SCC *that is a subgraph of Q_i and all nodes of the may-SCC are uncolored at the beginning of phase 2b.*

Proof: First note that any node that all its sons are already colored in phase 1 or phase 2a, gets colored as well in these phases. This is clear for phase 1 by the description of the coloring algorithm, since all the cases where all the sons are colored are handled. As for phase 2a, the reasoning is similar, with the exception that some of the cases where all the sons of a node are colored by *definite* colors (T or F) are *not* handled. Yet, if all the sons of a node are colored by definite colors in this phase, this means that they were already colored in phase 1 (since in phase 2a nodes get colored by ? only), which would make their parent already colored as well. That is, cases where all the sons are colored by definite colors are not possible in phase 2a.

Thus, each node n' that is uncolored at the beginning of phase 2b has an uncolored son n''. This is clear because otherwise n' would be colored as well either in phase 1 or in phase 2a. Since all the sons of n' are either in the same set Q_i or in smaller sets which are already colored, the uncolored son n'' is definitely within Q_i as well. Since Q_i is finite, this results in a non-trivial SCC, which is a subgraph of Q_i and is uncolored at the beginning of phase 2b.

It remains to refer to the type of the resulting SCC (*must* versus *may*). Clearly, it is a may-SCC (since a must-SCC is also a may-SCC). Thus, for the case of an EU or an AV witness, the claim is implied. We now consider the case of an AU or an EV witness. To do so, we refer to the type of edges that connect an uncolored node n' to its son n'' within the SCC in this case. For any node n' in Q_i other than AX-nodes or EX-nodes the connecting edge is an auxiliary edge. As for AX-nodes (in a set with an AU witness) and EX-nodes (in a set with an EV witness), at least one of the uncolored sons has to be a must-son. This is because if all the must-sons were colored, then by the description of the algorithm, n' would already be colored by previous phases (either by phase 1 if at least one of them is colored F for AX or T for EX, or by phase 2a otherwise). Thus, in this case we choose n'' to be an

uncolored *must-son* of n'. By previous arguments, we get an uncolored non-trivial SCC whose progress edges are must-edges. i.e. this is a *must-SCC*, as required.

Either way, we get that a node n that is uncolored at the beginning of phase 2b lies on a non-trivial SCC, with the required type, that is a subgraph of its Q_i and all nodes of the SCC are uncolored at the beginning of phase 2b. □

Proof of Theorem 4.5: The proof is by induction on the computation of the coloring algorithm. It suffices to prove the implication from each result of the coloring of n to the existence (or non-existence) of winning strategies.

Base case: n is a terminal node. On the one hand, by the coloring algorithm, n is colored according to the player that wins the game in such a configuration. On the other hand, this also determines the existence of a winning strategy, since each play that starts from such a configuration also ends in it. Thus, the claim is implied.

Induction step:

1. n is colored due to the coloring of its sons in phase 1.

   Consider the case where n is an AX-node. The first move in each play that starts from the configuration n is done by ∀belard.

   - If n is colored T then by the description of the algorithm all its may-sons are colored T, which means ∃loise has a winning strategy from each one of them (by the induction hypothesis). Thus, no matter which move ∀belard makes (this is his turn), she can win, i.e. she has a winning strategy from n, which is the union of the winning strategies of all the sons of n.
   - If n is colored F then by the description of the algorithm it has a must-son n' that is colored F, which means ∀belard has a winning strategy from n' (by the induction hypothesis). Thus, ∀belard has a winning strategy from n, which is to choose n' as the first move and continue by the guaranteed winning strategy from n'. Note, that the choice of n' provides a false-consistent play, since it is a must-son of n.
   - If n is colored by ? then by the description of the algorithm, it has a may-son n' that is colored by ? or F, and all its must-sons are colored by T or ?. The existence of n' assures us that ∃loise does not have a winning strategy for a game that starts from n, since ∀belard (which makes the move in such a configuration) can always choose as his first move the configuration n', for which ∃loise does not have a winning strategy by the induction hypothesis. In addition, the information about the must sons, assures us that ∀belard does not have a winning strategy from n, since for the play to be false-consistent ∀belard has to proceed to one of the must-sons, but from them, by the induction hypothesis, he does not have a winning strategy.

If n is an $\wedge$-node, other than AX-node, then the same proof holds, where instead of may or must edges we use auxiliary edges.

If n is an EX-node, or another $\vee$-node, then the proof is symmetric to the proof for AX or $\wedge$-nodes respectively, where the coloring rules are opposite and so are the roles of the players.

2. n is colored due to a witness during phase 2 in a Q_i that is a non-trivial may-MSCC (by Lemma 4.3).

   Consider the case where the witness is $A(\varphi_1 U \varphi_2)$ or $E(\varphi_1 U \varphi_2)$. Then n is colored by either ? (in phase 2a) or F (in phase 2b).

   We first prove that either way, $\exists$loise does not have a winning strategy for the game that starts from n. Let B be the *maximal* subgraph of $G_{M \times \varphi}$ that is reachable from n through uncolored nodes by the time phase 2 (coloring by witness) starts. Obviously, B is a subgraph of Q_i (because by the choice of $\leq$, only nodes in Q_i or smaller sets are reachable from n and the ones in smaller sets are already colored). Since it is a subgraph of Q_i, then by Lemma 2.6, its formulae are from $exp(A(\varphi_1 U \varphi_2))$ or $exp(E(\varphi_1 U \varphi_2))$.

   Suppose that $\exists$loise has a winning strategy for the game that starts from n. Then $\exists$loise manages to "force" the play to exit B to a configuration for which she has a winning strategy. This is because if the play stays within B, then the play is infinite (there are no terminal nodes in $exp(A(\varphi_1 U \varphi_2))$ or $exp(E(\varphi_1 U \varphi_2))$ thus the play cannot end without exiting B) and the witness is $A(\varphi_1 U \varphi_2)$ or $E(\varphi_1 U \varphi_2)$. Thus $\exists$loise loses.

   First, let us show that it is not possible that $\forall$belard is "forced" to exit B. Any node in B is uncolored at the beginning of phase 2 and thus lies on an uncolored non-trivial SCC (by Lemma 4.3). Hence, in particular, it has an uncolored son, which is also in B (by the maximality of B). Thus, for every node in B there exists a consecutive node that $\forall$belard can choose in his moves.

   Now, we show that $\exists$loise cannot exit B to a configuration for which she has a winning strategy, which contradicts the assumption that she has a winning strategy. The only configurations in which $\exists$loise makes a move are $\vee$-nodes, with subformulae of the form $\varphi' \vee \varphi''$ or $EX\varphi'$. Thus, if $\exists$loise manages to exit B, she does it in such a configuration. We consider each possibility separately.

   $\varphi' \vee \varphi''$: This means that there exists a configuration $(s', \varphi' \vee \varphi'')$ in B for which, without loss of generality, $\exists$loise chooses $n' = (s', \varphi')$, which is outside B, as her next move. This node, n', is already colored at the beginning of phase 1 (otherwise it would be in B as well, seeing that it is obviously reachable, since its parent is in B). In addition, by our assumption $\exists$loise has a winning strategy from n', thus by the induction hypothesis it is colored by T. However, by the coloring algorithm this means that the configuration $(s', \varphi' \vee \varphi'')$ that is *in* B could already be colored by T as well in phase 1, in contradiction to the property that B is uncolored.

$EX\varphi'$: If $\exists$loise exits B in a configuration of the form $(s', EX\varphi')$, then similar arguments apply, with the difference that in this case $\exists$loise chooses a *must*-son $n'' = (s'', \varphi')$ outside B as her next move. It has to be a *must*-son since this move is a part of her winning strategy, thus it has to be based on a must-transition from s' to s'' (the play needs to be true-consistent for her to win). By the same arguments, this configuration (node) n'' is already colored. In addition, $\exists$loise has a winning strategy from n'', thus by the induction hypothesis it is already colored by T. Again, this results in contradiction, since by the coloring algorithm this would cause its parent that is *in* B to be colored by T as well in phase 1.

We now show that if n is colored by ? (in phase 2a), then none of the players has a winning strategy from it, and if it is colored by F (in phase 2b), then $\forall$belard has a winning strategy.

(a) Consider the case where n is colored in phase 2a. i.e., it is colored by ?. In this case, it remains to show that $\forall$belard does not have a winning strategy for the game that starts from n (We already know this about $\exists$loise). The proof is by induction on the computation of phase 2a, where the main idea is that a node n is colored by ? only when the F option is overruled.

- If n is an AX-node (or an $\wedge$-node), this means that all its must-sons are colored by ? or T, such that indeed $\forall$belard has no winning strategy from this node: he can either choose a may-son, which will make the play not false-consistent, or he can choose a must-son, for which he has no winning strategy, by the induction hypothesis.
- If n is an EX-node (or an $\vee$-node), this means that it has a may-son n' that is colored by ? or T, such that indeed $\forall$belard has no winning strategy from this node: in such a configuration $\exists$loise makes the first move, such that she can choose n' for which $\forall$belard has no winning strategy by the induction hypothesis.

(b) Consider the case where n is colored in phase 2b. i.e., it is colored by F. We show that $\forall$belard has a winning strategy for the game from n.

$A(\varphi_1 U \varphi_2)$: Let B be the *maximal* uncolored subgraph of $G_{M\times\varphi}$ that is reachable from n through uncolored nodes and through must-edges or auxiliary edges (but *not* may-edges) by the time phase 2b of the coloring algorithm starts. B is clearly a subgraph of Q_i with formulae from $exp(A(\varphi_1 U \varphi_2))$.

The winning strategy of $\forall$belard is to always stay in B. Since the only progress edges in B are *must-edges*, then by this strategy $\forall$belard maintains the play false-consistent. Therefore, if he manages to stay in B, he wins (we get an infinite play with an AU-witness, which is also false-consistent).

First, we show that it is not possible that $\forall$belard is "forced" to exit B. By the choice of B, every node $n' \in B$ is uncolored at the beginning

of phase 2b, thus by Lemma 4.6 it lies on a non-trivial must-SCC, which is a subgraph of Q_i that none of its nodes are colored by the time phase 2b starts. Note, that whereas Q_i is a *may*-MSCC, we claim that n' lies on an uncolored *must*-SCC. In particular, it has an uncolored son, connected to it either by an auxiliary edge or by a must-edge, which is also in B (by the maximality of B). Thus, for every node in B there exists a consecutive node that ∀belard can choose in his moves.

We now consider the case where ∃loise exits B and show that in this case ∀belard still has a strategy that allows him to win (which will be added to his winning strategy). The only configurations in which ∃loise makes a move in such an SCC are of the form $n' = (s', \varphi' \vee \varphi'')$. Thus, the only way for ∃loise to exit B, is by choosing to proceed to $n'' = (s', \varphi')$ (without loss of generality) in such a configuration in B. By our assumption, n'' is not in B and thus already colored (it is connected to $n' \in B$ by an auxiliary edge, thus if it was not colored, it would be in B). It must be colored by F, otherwise n' would be already colored in phase 2a (as an $\vee$-node that has a son colored by T or ?) or before, in contradiction to its being a node in B. Since n'' must be colored by F, we get that ∀belard has a way to win the game even if ∃loise forces him to exit B (by the induction hypothesis).

$E(\varphi_1 U \varphi_2)$**:** Let B be the *maximal* subgraph that is reachable from n through uncolored nodes by the time phase 2b of the coloring algorithm starts. B is obviously a subgraph of Q_i, thus the formulae in B are formulae from $exp(E(\varphi_1 U \varphi_2))$.

The winning strategy of ∀belard is to always stay within B. This is a winning strategy because if he manages to stay within B, we get an infinite play with a witness $E(\varphi_1 U \varphi_2)$, which is also false-consistent (in fact ∀belard never uses any progress edges), thus he wins.

Obviously, ∀belard can always stay within B in his moves, since any node in B is uncolored at the beginning of phase 2b and thus lies on an uncolored non-trivial SCC (by Lemma 4.6) and in particular has an uncolored son which is also in B.

Furthermore, if ∃loise exits B, then her move is made in an $\vee$-node (possibly an EX-node), and she reaches a configuration that is already colored (otherwise it would be in B, by the choice of B and its maximality), and thus it is colored by F (otherwise its parent would be colored during phase 2a or before, in contradiction to its being in B). Hence, by the induction hypothesis, ∀belard has a winning strategy from this configuration as well.

If the witness is of the form $A(\varphi_1 V \varphi_2)$ or $E(\varphi_1 V \varphi_2)$, then the proof is symmetric, where the coloring rules are opposite and so are the roles of the players.

□

Implementation issues: First, note that the correctness of the coloring algorithm is not damaged if during phase 1 of the ith iteration, nodes from sets other than Q_i are colored as well. This results from the property that once a node is colored, its color never changes. In other words, coloring successors of an already colored node does not change its color. Thus, if a node from a set Q_j can be colored in phase 1 of the ith iteration ($i < j$) based on its sons, then these sons will still be colored the same in the jth iteration, leading to the same coloring (although possibly additional sons will be colored in the jth iteration).

Based on this observation, the coloring algorithm can be implemented in linear running time, using an AND/OR graph, similarly to the algorithm described in [29] for checking the nonemptiness of the language of a simple weak alternating word automaton. The algorithm maintains an integer i that contains the current iteration and three stacks S_T, S_F and $S_?$. The stacks contain nodes that were colored by T, F or ?, yet still have not propagated their coloring further. At the beginning, the stacks are initialized by all the terminal nodes, according to their coloring. During the son-based coloring (phase 1), whenever a node is colored, it is pushed into one of the stacks, based on its color. As long as the stacks are not empty, a node is popped from one of them and its parents are checked to see if they can be colored (based on *all* their sons).

When the stacks are empty, the witness-based coloring (phase 2) is applied on the smallest Q_i that contains uncolored nodes. First, phase 2a is applied, where initially all the nodes in Q_i are checked once to see if they can be colored. The ones that are colored are pushed into $S_?$ as well as a temporary stack. The temporary stack is used to propagate their coloring within Q_i based on the rules of phase 2a and each node that gets colored is added to the temporary stack and to $S_?$. Phase 2a ends when the temporary stack is empty. At that time, all the remaining nodes in Q_i are colored by T or F (depending on the witness) in phase 2b and are pushed into the appropriate stack. The algorithm then returns to phase 1 (with the new content of the stacks).

Complexity: The running time of the coloring algorithm is linear with respect to the size of the game-graph $G_{M\times\varphi}$. The latter is again bounded by the size of the underlying KMTS times the length of the CTL formula, i.e. $O(|M| \cdot |\varphi|)$.

As a conclusion of Theorem 4.2 and Theorem 4.5, we get the following theorem.

Theorem 4.7 *Let M be a KMTS and φ a CTL formula. Then, for each $n = (s, \varphi_1) \in G_{M\times\varphi}$:*

1. $[(M,s) \models^3 \varphi_1] = tt$ *iff* $n = (s,\varphi_1)$ *is colored by* T.
2. $[(M,s) \models^3 \varphi_1] = ff$ *iff* $n = (s,\varphi_1)$ *is colored by* F.
3. $[(M,s) \models^3 \varphi_1] = \perp$ *iff* $n = (s,\varphi_1)$ *is colored by* ?.

Given the colored game-graph, if all the initial nodes are colored by T, or if at least one of them is colored by F, then by Theorem 4.7 along with Theorem 2.14, there is a definite answer as for the satisfaction of φ in the *concrete* model. This is because there exists a mixed simulation from the concrete model to the abstract one. Furthermore, if the result is ff, a *concrete* annotated counterexample can be produced, using an extension of the `ComputeCounter` algorithm as described in the following chapter.

Example 4.8 Figure 4.1 presents a 3-valued game-graph G and its coloring, where dashed edges represent may-edges and solid edges represent must-edges, as well as auxiliary edges. The partition of G to Q_i's (may-MSCCs) is depicted by rectangles in Figure 4.1, where their indices 1-5 determine the order $\leq$. The coloring algorithm starts from Q_1-Q_4, where the terminal nodes are colored in phase 1 of the coloring algorithm. It then handles Q_5. $(s_1, p \wedge AXA(pUq))$ is colored F in phase 1 due to its son (s_1, p) and $(s_1, q \vee (p \wedge AXA(pUq)))$ is colored T due to its son (s_1, q), causing $(s_1, A(pUq))$ to be colored T as well. At this point, none of the remaining nodes can be colored in phase 1. Thus, the algorithm proceeds to phase 2, with Q_5 containing an AU-witness. The node $(s_1, AXA(pUq))$ is colored ? in phase 2a, since it is an AX node and its only must son is colored T. The rest of the nodes are then colored F in phase 2b. This example demonstrates that if a node is left uncolored after phase 2a in a set with an AU-witness, then it lies on a non-trivial must-SCC that provides an evidence for refutation. The final coloring function can be seen in Figure 4.1, where white nodes are colored T, dark grey nodes are colored F and light grey nodes are colored ?. Based on Theorem 4.7, that describes the relation between the coloring results and model checking, we can use the color of each node in order to conclude what is the truth value of its formula in its state. In particular, since the initial node $(s, A(pUq))$ is colored by F, we can conclude that the value of the formula $\varphi = A(pUq)$ in the initial state s of the model M is ff. This is indeed correct since the value of the until formula pUq in the (infinite) must-path $s, s, \ldots$ is ff (the value of p is always ff in this path, thus the eventuality of the until is definitely refuted), and the existence of such a must-path suffices to definitely refute the universal formula $A(pUq)$, resulting in the truth value ff. Since s is an initial state of the model M, we can conclude that the value of $\varphi = A(pUq)$ in the abstract model M is ff. This implies that φ is refuted in the corresponding concrete model, represented by M.

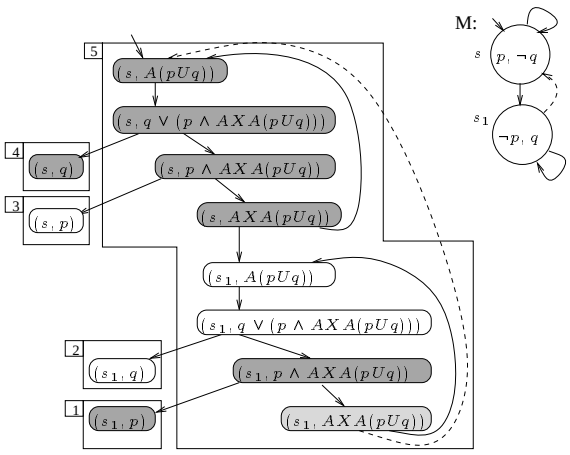

Figure 4.1: A colored 3-valued game-graph for M and $\varphi = A(pUq)$, where dashed edges are may-edges, solid edges are must-edges or auxiliary edges, and rectangles depict the partition of the nodes. White nodes are colored by T, dark grey nodes are colored by F, and light grey nodes are colored by ?.

Chapter 5

Concrete Annotated Counterexamples Based on Abstract Game-Graphs

In this chapter we show how to produce a *concrete* annotated counterexample from the 3-valued abstract game-graph that was used for model checking, in case that one of the initial nodes was colored by F, meaning that $[M \models \varphi] = \text{ff}$.

Let $M_C = (S_C, S_{0C}, \rightarrow, L_C)$ be a (concrete) Kripke structure and let $M_A = (S_A, S_{0A}, \xrightarrow{\text{must}}, \xrightarrow{\text{may}}, L_A)$ be a (abstract) KMTS as described in the preliminaries, such that $M_C \preceq M_A$, where $\preceq$ is the mixed simulation relation. Let $\gamma : S_A \rightarrow 2^{S_C}$ be the concretization function. Given the abstract 3-valued game-graph G_A, based on φ and the abstract model M_A, and its coloring function $\chi : N \rightarrow \{T, F, ?\}$, such that $\chi(n_{0a}) = F$ for some initial node $n_{0a} = (s_{0a}, \varphi)$, we first use a variation of the previously described `ComputeCounter` to produce an abstract annotated counterexample C_A. The difference in the abstract version of the algorithm is that for an AX-node, where the *cause* is added to the annotated counterexample, we now choose the cause to be a *must-son*, and in an EX-node, we add all its *may-sons* to the annotated counterexample.

In order to find a concrete annotated counterexample out of C_A, we then need to replace each abstract state s_a in C_A, that represents a set of concrete states, with a single concrete state s_c from $\gamma(s_a)$. Since we are dealing with an annotated counterexample, some of the edges between nodes are auxiliary edges, that do not represent advancements along transitions of the model. If this is the case then the same concrete state should eventually match both these nodes. For an AX-node, the annotated counterexample shows one son that refutes the property. Given such a node n_a, and its only son in the counterexample n'_a, we need to match both their states with concrete states that have a concrete transition between them. For an EX-node, the annotated counterexample shows refutation in all its sons. Hence, given such a node n_a, we need to match its abstract state s_a with a concrete state s_c and add *all* its concrete sons to the concrete annotated counterexample.

Hence, the **concretization algorithm** of C_A for producing a concrete annotated counterexample, C_C, is described as follows.

- Choose the initial concrete node to be $n_{0c} = (s_{0c}, \varphi)$, where s_{0a} is the initial abstract state that appears in n_{0a} and s_{0c} is an arbitrary node from $\gamma(s_{0a}) \cap S_{0C}$.
- Apply the recursive procedure `ComputeSons` on $(n_{0c},\ n_{0a})$.

Given a concrete node $n_c = (s_c, \varphi')$ and the abstract node $n_a = (s_a, \varphi')$ that matches it, the procedure `ComputeSons`(n_c, n_a) creates the concrete sons of n_c as follows:

- If $\varphi' = EX\varphi_1$, then for each state s'_c such that $s_c \rightarrow s'_c$, the node (s'_c, φ_1) is added to the concrete annotated counterexample as a son of n_c. Each such node matches an abstract node $n'_a = (s'_a, \varphi_1)$, such that $s'_c \in \gamma(s'_a)$. Moreover, n'_a is a son of n_a in the abstract annotated counterexample.
- If $\varphi' = AX\varphi_1$, then n_a has one son $n'_a = (s'_a, \varphi_1)$ in C_A. An arbitrary state s'_c is chosen from $\{s'_c \in S_C : s_c \rightarrow s'_c\} \cap \gamma(s'_a)$ and the node (s'_c, φ_1) is added to the concrete annotated counterexample as a son of n_c. The resulting son matches n'_a.
- If $\varphi' = \varphi_1 \vee \varphi_2$, then the nodes (s_c, φ_1) and (s_c, φ_2) are added to the concrete annotated counterexample as sons of n_c. They match the abstract nodes (s_a, φ_1) and (s_a, φ_2) respectively, which are both sons of n_a in C_A.
- If $\varphi' = \varphi_1 \wedge \varphi_2$, then n_a has one son $n'_a = (s_a, \varphi_i)$ in C_A, where $i \in \{1, 2\}$. The node (s_c, φ_i) is added to the concrete annotated counterexample as a son of n_c. It matches the abstract node n'_a.

In any case, we then recursively call the procedure `ComputeSons` on the *new* concrete nodes (each one and the abstract node that it matches).

Basically, this is a greedy algorithm. The only situation where there is "freedom" in the choice of concrete states is in case of sons of AX-nodes. In EX-nodes the algorithm makes sure to include all the concrete sons in the annotated counterexample. As for other nodes, whose sons result from auxiliary edges, the algorithm makes sure to attach both the parent and the son with the same state.

Complexity: The running time of the concretization algorithm is linear in the size of the concrete annotated counterexample, which is bounded by the size of the concrete Kripke structure M_C times the length of the CTL formula φ, i.e. $O(|M_C| \cdot |\varphi|)$.

Lemma 5.1 *The concretization algorithm does not fail.*

Proof: This results from the following properties of the mixed simulation between M_A and M_C, induced by γ, and of the abstract annotated counterexample.

1. For each initial abstract state s_{0a}, there exists $s_{0c} \in S_{0C}$ such that $s_{0c} \in \gamma(s_{0a})$ (since $M_C \preceq M_A$). Thus we have that $\gamma(s_{0a}) \cap S_{0C} \neq \emptyset$.

2. If $s_c \in \gamma(s_a)$, then for each s'_c such that $s_c \rightarrow s'_c$, there exists an abstract state s'_a such that $s'_c \in \gamma(s'_a)$ and $s_a \xrightarrow{\text{may}} s'_a$. In addition, for EX-nodes the abstract annotated counterexample C_A contains *all* the may-sons. Thus, in the concretization of an EX-node, we are guaranteed that all its concrete sons are represented by the sons of its matching node in C_A.

3. If $s_a \xrightarrow{\text{must}} s'_a$, then for each $s_c \in \gamma(s_a)$ there exists $s'_c \in \gamma(s'_a)$ such that $s_c \rightarrow s'_c$, thus $\{s'_c \in S_C : s_c \rightarrow s'_c\} \cap \gamma(s'_a) \neq \emptyset$. In addition, for AX-nodes the abstract annotated counterexample C_A contains a *must-son*. Thus, in the concretization of an AX-node, we are guaranteed that there exists a concrete son that is represented by the abstract son in C_A.

□

In order to prove the correctness of the concretization algorithm, we need to show that the result C_C that it produces is indeed a (concrete) annotated counterexample for the concrete game-graph $G_C = (N_C, E_C)$, based on M_C and φ. To do so, we need to show that C_C is a subgraph of G_C that is (1) colored F by the coloring function of G_C, denoted $\chi : N_C \rightarrow \{T, F\}$, (2) independent of G_C, i.e. that every partial coloring function of G_C w.r.t C_C does not change the colors of its nodes, and (3) minimal.

Lemma 5.2 *C_C is a subgraph of G_C.*

Proof: This results from the following properties.

- The initial node consists of a concrete initial state and the original formula φ (which also appeared in the abstract initial node).

- Every other node is a legal son of its parent. The correctness in terms of the formulae in the nodes results from the abstract annotated counterexample, because the formulae are not changed. As for the states, if the parent is an AX or EX node, the correctness results from the fact that we replaced the may or must edges with concrete edges, such that there exists a real transition in the model from the parent's state to its son's state. Otherwise, it results from the fact that we made sure to have the same state both in the parent and in its son.

□

As for (1) and (2), it suffices to show that both the regular coloring algorithm and the partial coloring algorithm w.r.t C_C, given any initial coloring function of $N_C \setminus C_C$, color the nodes of C_C by F. In fact, the regular coloring algorithm can be viewed as an instance of the partial coloring algorithm with respect to C_C where the initial coloring function of $N_C \setminus C_C$ colors each node $n \in N_C \setminus C_C$ by $\chi(n)$. Thus, it suffices to prove this claim for the partial coloring algorithm only.

Lemma 5.3 *Let n_c be a node in C_C that corresponds to a node n_a in C_A. Then given any initial coloring function of $N_C \setminus C_C$, n_c is colored by F in the partial coloring of G_C w.r.t C_C.*

Proof: Given any initial coloring function that colors $N_C \setminus C_C$, the proof is by induction on the computation of the partial coloring algorithm w.r.t C_C. We consider nodes in C_C only. The rest are colored by the initial coloring function, however their colors are not relevant to the proof. Note that each node $n_c \in C_C$ has exactly one matching node in C_A, denoted by n_a (by the concretization algorithm).

Base case: $n_c \in C_C$ is a terminal node, thus so is its matching abstract node n_a, because they have the same formula. Since $n_a \in C_A$, we have that it was colored by F. Thus, their formula can be either false or l where $\neg l \in L_A(s_a)$. In the first case, n_c is obviously colored by F. In the second case, by the concretization algorithm, we have that the (concrete) state s_c of n_c belongs to $\gamma(s_a)$. In addition, $\neg l \in L_A(s_a)$ means that $\neg l$ labels *all* the concrete states in $\gamma(s_a)$ and in particular $\neg l \in L_C(s_c)$. Hence, n_c is colored by F as well.

Induction step:

1. $n_c \in C_C$ is colored due to its sons. We prove that it is colored by F. Assume to the contrary that n_c was colored by T based on its sons.

   - If n_c is an $\vee$-node, this means that it has a son $n'_c \in G_C$ that is already colored by T. Clearly, by the description of the concretization algorithm, we get that all of the sons of n_c from G_C appear in C_C. This is true in particular for n'_c. Therefore, by the induction hypothesis, if it is already colored, then it must be colored by F, in contradiction.
   - If n_c is an $\wedge$-node, this means that all its sons in G_C are already colored by T. By the concretization algorithm, one of its sons, denoted by n'_c, is in C_C. However, by our assumption, n'_c is already colored by T, which contradicts the Induction hypothesis.

2. n_c was colored due to witness. It suffices to show that the witness can not be AV or EV. Similarly to Lemma 3.11, it can be shown that for n_c to be colored by a witness, it must reside on a loop of nodes that matches an abstract loop that passes through n_a in C_A. However, by a generalization of Lemma 3.3 to the abstract case, if n_a resides on a loop in C_A, then at least one of the nodes n'_a on the loop must have been colored by a witness. Now, if the witness that caused the coloring of n_c was AV or EV, then so would be the witness that caused the coloring of n'_a (because n'_a lies on a loop that contains n_a, thus by Lemma 2.6 which holds for the abstract case as well, their formulae result from the same witness and the formulae in n_c and n_a are the same), which means n'_a would be colored by T, in contradiction to its being in C_A.

□

Lemma 5.4 *C_C is minimal in the sense that any node and edge that are removed from it result in a subgraph of G_C that is not independent of G_C.*

Proof: similar to the proof of Theorem 3.12. □

We can now conclude the correctness of the concretization algorithm, guaranteed by the following Theorem.

Theorem 5.5 *C_C is an annotated counterexample for M_C and φ.*

Chapter 6

Refinement

In this chapter, we show how to exploit the abstract game-graph in order to refine the abstract model in case that the model checking resulted in an indefinite answer.

In the framework of abstraction-refinement, refinement is usually based on a spurious counterexample, and the state in which its concretization fails. This approach is suitable when the model checking is based on 2-valued semantics, where tt is definite, whereas ff is not, such that an abstract counterexample may be spurious. The goal of the refinement is then to eliminate the spurious counterexample.

When dealing with the 3-valued semantics, if the result of the model checking is ff, then it is definite as well and no refinement is needed. In this case, refinement is needed when the resulting truth value is indefinite, i.e. $\perp$, in which case there is no reason to assume either one of the definite answers tt or ff. Thus, we would like to base the refinement not on a counterexample as in [30, 9, 3, 12, 8], but on the point(s) that are responsible for the indefinite answer. The goal of the refinement is to discard these points, in the hope of getting a definite result on the refined abstraction.

Let $M_C = (S_C, S_{0C}, \rightarrow, L_C)$ be a concrete Kripke structure and let $M_A = (S_A, S_{0A}, \xrightarrow{\text{must}}, \xrightarrow{\text{may}}, L_A)$ be an abstract KMTS as described in the preliminaries such that $M_C \preceq M_A$. Let $\gamma : S_A \rightarrow 2^{S_C}$ be the concretization function. Given the abstract 3-valued game-graph G, based on the abstract model M_A and its coloring function $\chi : N \rightarrow \{T, F, ?\}$, such that $\chi(n_0) =?$ for some initial node n_0, we use the information gained by the coloring algorithm of G in order to refine the abstraction. We refine the abstract model by splitting its abstract states according to criteria obtained by a *failure node*.

Definition 6.1 *A node n is a* failure node *if it is colored by ?, whereas none of its sons was colored by ? at the time n got colored by the coloring algorithm.*

Informally, such a node is a failure node in the sense that it can be seen as the point where the loss of information occurred. Thus, it can guide the refinement in hope to avoid the loss of information.

Note that a failure node may have uncolored sons at the time it is colored, some of which may eventually be colored by ?. Also note, that a terminal node that is colored by ? is also considered a failure node.

6.1 Finding a Failure Node

First, the coloring algorithm is adapted to identify and remember failure nodes. In addition, for each node n that is colored by ?, but is *not* a failure node, the coloring algorithm remembers a son that was already colored ? by the time n was colored, denoted $cont(n)$. Now, given the initial node n_0 that is colored ? a failure node is found by the following DFS-like greedy algorithm, starting from n_0.

Algorithm `FailureSearch`

- If the current node n is a *failure node*, the algorithm ends and returns it.
- As long as n is not a failure node, the algorithm proceeds to $cont(n)$ (recursively).

Lemma 6.2 *The algorithm* `FailureSearch` *terminates.*

Proof: As long as the current node n is not a failure node, the algorithm continues to $cont(n)$. This is a node that was colored ? *prior* to n. Note, that by the definition of a failure node, there always exists such a node if n is not a failure node. Thus, each recursive call is applied on a node that was colored ? earlier. Hence, the number of recursive calls is bounded by the running time of the coloring algorithm, which is finite. □

Lemma 6.3 *A failure node, and in particular the one returned by the algorithm* `FailureSearch`*, is a node colored by ?, which is one of the following.*

1. *A terminal node of the form (s_a, l) where $l \in Lit$.*
2. *An AX-node (EX-node) that has a may-son colored by F (T).*
3. *An AX-node (EX-node) that was colored during phase 2a based on an AU (EV) witness, and has a may-son colored by ?.*

Proof: By its definition, a failure node n is colored by ? and none of its sons were colored by ? at the time it got colored. Thus, it cannot be an $\wedge$-node or an $\vee$-node other than an AX-node or an EX-node. This is because such a node can never be colored by ? when none of its sons are colored by ?. More specifically:

1. This is clear from the description of the coloring algorithm when n is colored during phase 1.

2. As for phase 2, n could get colored by ? only in phase 2a. If the witness is an AU or an EU witness, then by the description of the coloring algorithm, an $\wedge$-node gets colored by ? only if both its sons are colored by T or ?. It is not possible that both of them were colored by T, since this would mean they were already colored this way in phase 1, thus n would already be colored in this phase as well. Similarly, an $\vee$-node gets colored by ? in this phase and with such a witness only if it has a son that is colored by T or ?. The T option is not possible since it would again mean that n could already be colored in phase 1. In any case, we get that an $\wedge$-node or an $\vee$-node that gets colored in phase 2a based on such a witness has a son that is already colored by ? at this time. Similar arguments apply to the case of an AV or an EV witness.

Thus, by definition, the failure node cannot be such a node, which means it can only be a terminal node, an AX-node or an EX-node.

- If n is a terminal node (case 1), then it must be of the form (s_a, l) since terminal nodes of the form (s_a, true), (s_a, false) are colored by definite colors rather than ?.

- Consider the case where the failure node n is an AX-node. We prove that either it has a may-son colored by F (case 2), or it was colored during phase 2a based on an AU witness, and has a may-son colored by ? (case 3). If n has a may-son colored by F, then we are done.

  If n does not have a may-son colored by F, then all its may-sons are clearly colored by T or ?. It cannot be the case that all of them are colored by T, since then n would also be colored by T, in contradiction to its being a failure node. Thus, n has at least one may-son that is colored by ?. Note, that we claim that this son is colored by ? at the *end* of the coloring, but *not* at the time n got colored (otherwise n would not be a failure node). It remains to show that if this case occurs then n was colored during phase 2a based on an AU witness.

  Obviously, n could not be colored during phase 1, since if this was the case, then by the coloring algorithm, n would have a may-son n' that was already colored by F or ? when n got colored. By our assumption, none of its may-sons is colored by F at the end of the coloring and thus also at that time, which implies that n has a may-son that was already colored by ?, which contradicts its being a failure node.

  Furthermore, n could not be colored during phase 2b of any iteration, otherwise it would be colored by a definite color, in contradiction to its being a failure node.

  Thus, we conclude that in the case where n does not have a may-son that is colored by F, n was indeed colored during phase 2a. In addition, since it is an AX-node, it can *not* be a part of an MSCC whose witness is EU or EV (by Lemma 2.6). Thus, it remains to eliminate the possibility that the witness is AV. If n was colored during phase 2a based on an AV-witness, then by the

coloring algorithm, when it got colored, it had a may-son n' that was already colored by F or ?. By our assumption, the F option is not possible, which means n had a may-son that was already colored by ?, in contradiction to its being a failure node. Thus, we conclude that the witness is an AU-witness.

As a conclusion we get that either n has a may-son colored by F, or it has a may-son colored by ?, in which case it was colored during phase 2a based on an AU witness.

- The case where n is an EX-node is similar.

□

6.2 Failure Analysis

Given a failure node n, it provides us with criteria for the refinement. Based on this criteria, the refinement problem is reduced to the problem of separating sets of (concrete) states, which can be solved by known techniques, depending on the type of abstraction used. Examples of solutions for certain types of abstractions can be found in [12] (for abstraction based on invisible variables) and in [9] (for abstraction based on formulae clusters). The criterion for the separation depends on the type of n and is found by the following analysis.

1. $n = (s_a, l)$ is a terminal node. In this case, its indefinite color results from the fact that s_a represents both concrete states that are labeled by l as well as concrete states that are labeled by $\neg l$. In order to avoid the indefinite color in this node, we need to separate these types of concrete states. Hence, our goal is to separate $\gamma(s_a)$ to two sets $\{s_c \in \gamma(s_c) : l \in L_C(s_c)\}$ and $\{s_c \in \gamma(s_c) : \neg l \in L_C(s_c)\}$.

2. $n = (s_a, AX\varphi_1)$ with a may-son colored F, or $n = (s_a, EX\varphi_1)$ with a may-son colored T. Let K stand for F or T. We define $sons_K = \bigcup\{\gamma(s'_a) : (s'_a, \varphi_1)$ is a may-son of n colored $K\}$ and $conc_K = \gamma(s_a) \cap \{s_c \in S_C : \exists s'_c \in sons_K, s_c \to s'_c\}$. For the $AX\varphi_1$ case, $K = F$ and $conc_K$ is the set of all concrete states, represented by s_a, that definitely refute $AX\varphi_1$. For the $EX\varphi_1$ case, $K = T$ and $conc_K$ is the set of all concrete states, represented by s_a, that definitely satisfy $EX\varphi_1$. In both cases, our goal is to separate the sets $conc_K$ and $\gamma(s_a) \setminus conc_K$.

3. $n = (s_a, AX\varphi_1)$ or $n = (s_a, EX\varphi_1)$ was colored during phase 2a based on an AU or an EV witness (respectively), and has a may-son $n' = (s'_a, \varphi_1)$ colored by ?. Let $conc_? = \gamma(s_a) \cap \{s_c \in S_C : \exists s'_c \in \gamma(s'_a), s_c \to s'_c\}$ be the set of all concrete states, represented by s_a, that have a son represented by s'_a. Our goal is to separate the sets $conc_?$ and $\gamma(s_a) \setminus conc_?$.

It is possible that one of the sets obtained during the failure analysis is empty and provides no criterion for the split. Yet, new information can be gained from it as well. For example, consider case 2, where the failure node n is an AX-node. if $conc_F = \gamma(s_a)$, then in fact every state that is represented by s_a has a refuting son. Thus, n can be colored by F instead of its current indefinite color ?. If $conc_F = \emptyset$, then in fact none of the concrete states in $\gamma(s_a)$ has a transition to a concrete state that is represented by the F-colored may-sons of n. Thus, the (abstract) may-edges from n to such sons, which caused n to be colored by ?, can be removed: none of them represents concrete transitions. Either way, the game-graph can be recolored based on the new information, starting from the may-MSCC containing n. Similar arguments apply to the rest of the cases as well.

The intuition behind the criterion for the split that is derived from cases 1-2 is clear. Its purpose is to allow us to conclude definite results about (at least part) of the new abstract states obtained by the split of the failure node. These new definite results can be used within the framework of the incremental algorithm, suggested in the next chapter. We now consider case 3. Intuitively, in this case we know that by the time the failure node n got colored, its may-son n' that is colored by ? was not yet colored (otherwise n would not be a failure node). By the description of the coloring algorithm, we know that if n' was a must-son of n, then as long as it was uncolored, n would remain uncolored too and would eventually be colored in phase 2b by a definite color. Thus, our goal in this case is on the one hand to obtain a must edge between (parts of) n and n' and on the other hand remove the may-edge altogether between other parts of n and n'.

Example 6.4 Consider the game-graph described in Figure 6.1(a). Its initial node is colored by ?, thus refinement is needed. To understand which node is a failure node and to identify $cont(n)$ for a node n that is colored by ? but is not a failure node, we need to follow the coloring algorithm. All the nodes in $Q_1 - Q_4$ are terminal nodes and are colored accordingly by definite colors. As for Q_5, the nodes $n_6 = (t, p \wedge AXA(pUq))$, $n_5 = (t, q \vee (p \wedge AXA(pUq)))$, and $n_4 = (t, A(pUq))$ are colored in phase 1 by definite colors. The rest are colored in phase 2a by ? as follows. The nodes $n_3 = (s, AXA(pUq))$, $n_2 = (s, p \wedge AXA(pUq))$, $n_1 = (s, q \vee (p \wedge AXA(pUq)))$ and $n_0 = (s, A(pUq))$ are colored in this order, which makes n_3 a failure node, whereas n_2, n_1 and n_0 are not failure nodes and for them $cont(n_0) = n_1$, $cont(n_1) = n_2$ and $cont(n_2) = n_3$. The node $n_4 = (t, AXA(pUq))$ can be colored at any time during this phase. If it is colored before n_0, then it is also a failure node, otherwise $cont(n_4) = n_0$. Given this information, the algorithm `FailureSearch` is applied starting from the initial node, $n_0 = (s, A(pUq))$. It continues to n_1, from there to n_2 and then reaches the failure node $n_3 = (s, AXA(pUq))$.

Given the failure node n_3, failure analysis is applied. n_3 is an AX-node. It does not have a may-son colored by F, thus as guaranteed by Lemma 6.3, it has a may-son $n_0 = (s, A(pUq))$ that is colored by ? and it was colored in phase 2a based on an AU-witness. That is, n_3 corresponds to the third case. Thus, we compute the set $conc_? = \gamma(s) \cap \{s_c \in S_C : \exists s'_c \in \gamma(s), s_c \rightarrow s'_c\}$, which is the set of all concrete states

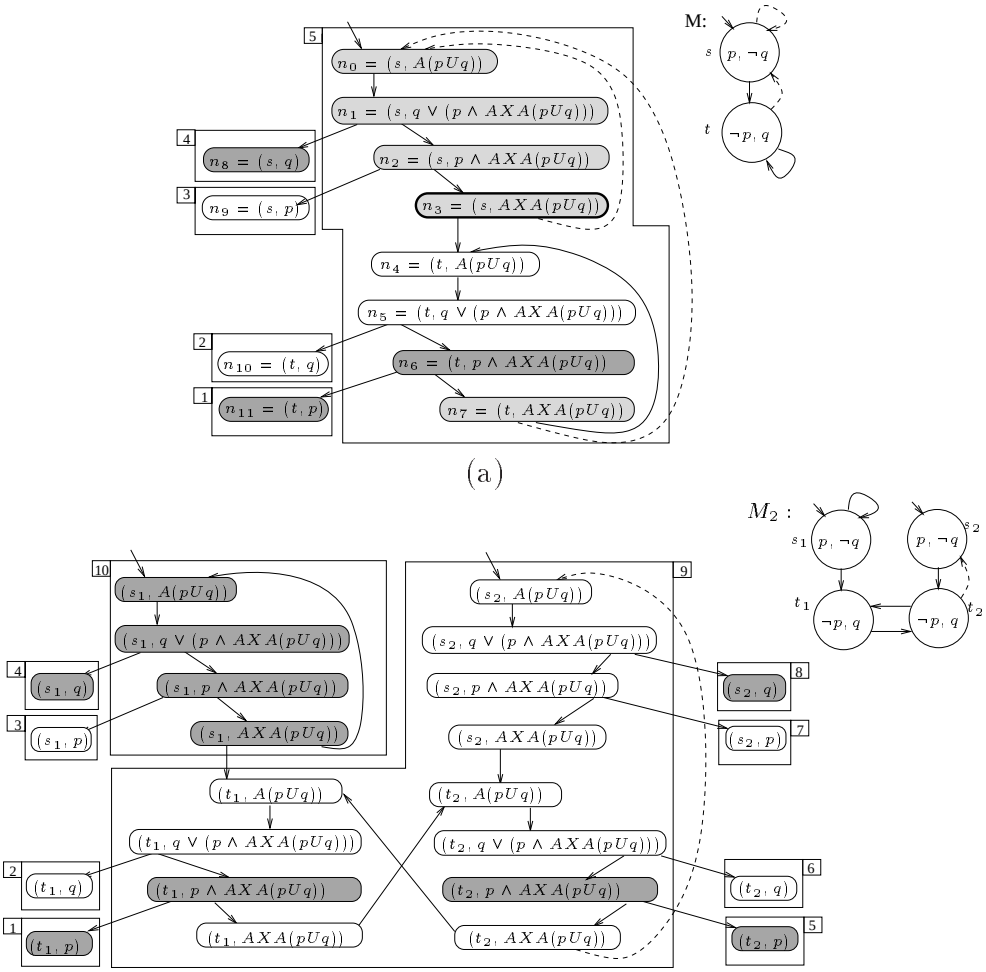

Figure 6.1: (a) A 3-valued colored game-graph for M and $\varphi = A(pUq)$, where the initial node is colored by ? and the failure node found by `FailureSearch` appears in boldface; (b) The colored refined game-graph, based on the refined model M_2 and φ.

represented by s that have a transition to a concrete state represented by s. The refinement is aimed at separating the sets $conc_?$ and $\gamma(s) \setminus conc_?$. For example, if the abstraction that is used is based on making some of the variables of the system *invisible*, then these sets may be separated by turning some of the invisible variables *visible* again.

Fig 6.1(b) presents a possible refined model M_2, where each abstract state was split into two abstract states (possibly due to the addition of one visible variable) and the abstract transitions were computed according to the new abstract states. It also presents the resulting refined game-graph, where the split in the abstract states caused the nodes of the game-graph to be split as well. For example, n_0 was split to $(s_1, A(pUq))$ and $(s_2, A(pUq))$. n_3 was split to $(s_1, AXA(pUq))$ and $(s_2, AXA(pUq))$, etc. It can be seen that in the refined game-graph the initial nodes are now colored by definite colors.

In this example the may-edge from the failure node n_3 to n_0 that existed in the game-graph described in Fig 6.1(a) and guided the refinement was indeed eliminated: it no longer exists as a may-edge in the refined game-graph described in Figure 6.1(b) between none of the nodes that resulted from n_3 and n_0. The nodes $(s_1, AXA(pUq))$ and $(s_1, A(pUq))$ that resulted from them now have a *must-edge* between them, whereas the node $(s_2, AXA(pUq))$, which also resulted from n_3 does not have *any* edge to the nodes $(s_1, A(pUq))$ and $(s_2, A(pUq))$ that resulted from n_0.

Note that in failure nodes of the type AX or EX we have ignored the information about sons that are colored by T or F respectively. This information may be used to derive criteria for further separation. For example, consider the case of an AX-node. Let $\widetilde{conc}_T = \gamma(s_a) \cap \overline{\{s_c \in S_C : \exists s'_c \in \overline{sons_T}, s_c \to s'_c\}}$ (where $\overline{A} \equiv S_C \setminus A$) be the set of all the concrete states represented by s_a, that *all* their (concrete) sons are represented by sons of n that are colored by T. These states all satisfy the AX formula. Thus, separating them from the rest may be helpful as well.

Theorem 6.5 *For finite concrete models, iterating the abstraction-refinement process is guaranteed to terminate with a definite answer.*

Proof: Applying the refinement on the abstract model results in an abstract model, whose states are more accurate in the sense that they represent (possibly) less concrete states. i.e., the refined model is "closer" to the concrete model than the original abstract model in terms of states. Thus, the number of iterations in the abstraction-refinement process is bounded by the number of concrete states and is guaranteed to end when the state space is finite. □

Chapter 7

Incremental Abstraction-Refinement Framework

We refine abstract models by splitting their states. The criterion for the refinement is decided locally, based on one node, but it has a global effect, since the refinement is applied to the whole abstract model. Yet, in practice, there is no reason to split states for which the model checking results are definite. The game-based model checking algorithm provides us with a convenient framework to use previous results and avoid unnecessary refinement. This leads to an *incremental* model checking algorithm based on iterative abstraction-refinement, where each iteration consists of abstraction, model checking and refinement. This chapter is devoted to the description of our incremental abstraction-refinement framework.

At the end of the ith iteration of abstraction-refinement, we now remember the (abstract) nodes that were colored by definite colors, as well as nodes for which a definite color was discovered during the failure analysis. Let D_i denote the set of such nodes. Let $\chi_{D_i} : D_i \to \{T, F\}$ be the coloring function that maps each node in D_i to its (definite) color. χ_{D_i} can be extracted from the result of the coloring algorithm.

At the jth iteration, let $D = \bigcup_{i<j} D_i$ denote the set of nodes that were remembered from previous runs, and let $\chi_D : D \to \{T, F\}$ denote their coloring function, $\chi_D = \bigcup_{i<j} \chi_{D_i}$.

During the construction of a new refined game-graph in the jth iteration, we prune the game-graph in nodes that are *sub-nodes* of nodes from D (nodes with definite colors from previous iterations). A node (s_a, φ) is a *sub-node* of (s'_a, φ') if (1) they both have the same subformula, i.e. $\varphi = \varphi'$, and (2) the set of concrete states represented by s_a is a subset of those represented by s'_a. When a node n that is a sub-node of a node $n_d \in D$ is encountered, we add $n_d \in D$ to the game-graph rather than n and do not continue to construct the game-graph from n, nor n_d. As a

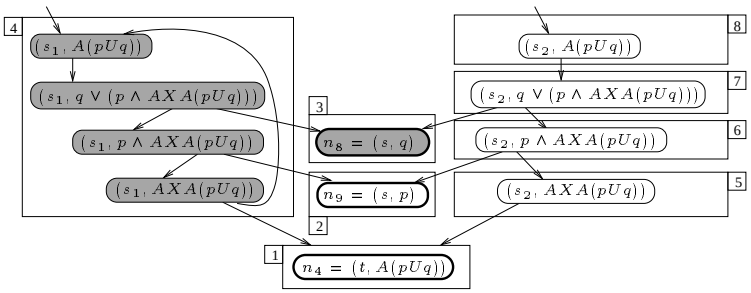

Figure 7.1: The pruned game-graph for M_2 and φ from Figure 6.1, where nodes from the initial game-graph, presented in Figure 6.1(a), appear in boldface.

result of this pruning, only the reachable subgraph that was previously colored by ? is refined.

The coloring algorithm then considers the nodes in D, where the game-graph was pruned, as terminal nodes and colors them by their previous colors, i.e. a node $n_d \in D$ is colored by $\chi_D(n_d)$. The rest of the algorithm remains unchanged. This is similar to the partial coloring algorithm, presented in Definition 3.6.

Note that for many abstractions, checking if a node is a sub-node of another is simple. For example, in the framework of predicate abstraction [23, 45, 40, 20], this means that the abstract states "agree" on all the predicates that exist before the refinement. When the abstraction is based on invisible variables [12], this means that the abstract states "agree" on all the variables that are visible before the refinement.

Example 7.1 Figure 7.1 demonstrates the use of previous results within the incremental algorithm, where after refining the model M described in Figure 6.1(a), instead of building the entire game-graph based on the refined model M_2, as described in Figure 6.1(b), the game-graph is pruned in $(t_1, A(pUq))$ and $(t_2, A(pUq))$ that are both sub-nodes of n_4 which already had a definite color (T). The same goes for (s_1, q) and (s_2, q) that are sub-nodes of n_8, and for (s_1, p) and (s_2, p) that are sub-nodes of n_9. The pruned game-graph, presented in Figure 7.1, is clearly smaller and simpler than the full refined game-graph. Its coloring handles the nodes in $Q_1 - Q_3$, where the game-graph was pruned, as terminal nodes and colors them by their previous colors. The nodes in Q_4 are all colored by F in phase 2b, based on the AU-witness, whereas the nodes in $Q_5 - Q_8$ are colored in phase 1.

Chapter 8

Conclusion

In this work, we have exploited the game-theoretic approach of CTL model checking to produce annotated counterexamples for full CTL. We have generalized this approach to 3-valued abstract models and suggested an incremental abstraction-refinement framework based on our generalization.

Traditional game-based model checking algorithms determine a winning strategy for the winning player. The winning strategy holds all the relevant information as for the result of the model checking, but it has redundancies. The annotated counterexample introduced in this thesis may be seen as a minimal part of it that is sufficient to explain the result.

Our 3-valued game-based model checking and in particular the failure nodes provide information for refinement, in case the outcome is indefinite. Additional information can be extracted from them and be used for further optimizations of the refinement.

The incremental abstraction-refinement algorithm described in this thesis can be viewed as a generalization of Lazy abstraction [26], which allows different parts of the abstract model to exhibit different degrees of abstraction. Lazy abstraction refers to safety properties, whereas our approach is applicable to full CTL.

This work is based on the game-theoretic approach to model checking. This approach is closely related to the Automata-theoretic approach [29], as described in [36]. Thus, our work can also be described in this framework, using alternating automata. In addition, it can easily be extended to alternation-free μ-calculus.

Appendix A

Discussion: 2-Valued Game-Based Model Checking

In our discussion on abstract models, we have used the 3-valued semantics for the interpretation of a CTL formula over a KMTS. The definition of $[(M,s) \models \varphi]$ can be extended to a KMTS using a *2-valued semantics* as well [16], where the possible truth values are tt and ff. The definition is similar to the concrete semantics with the following changes. Universal properties, of the form, $A\psi$, are interpreted along *may* paths. Existential properties, of the form $E\psi$, are interpreted along *must* paths. This gives us the *2-valued semantics* of CTL formulae over KMTSs, denoted $[(M,s) \stackrel{2}{\models} \varphi] = \mathrm{tt} \; / = \mathrm{ff}$. The 2-valued semantics is designed to preserve the truth of a formula from the abstract model to the concrete one. However, false alarms are possible, where the abstract model falsifies the property, but the concrete one does not.

Theorem A.1 *[16] Let $H \subseteq S_C \times S_A$ be a mixed simulation relation from a Kripke structure M_C to a KMTS M_A. Then for every $(s_c, s_a) \in H$ and every CTL formula φ, we have that $[(M_A, s_a) \stackrel{2}{\models} \varphi] = tt$ implies that $[(M_C, s_c) \stackrel{2}{\models} \varphi] = tt$.*

We conclude that if $M_C \preceq M_A$, then $[M_A \stackrel{2}{\models} \varphi] = tt$ implies that $[M_C \stackrel{2}{\models} \varphi] = tt$.

The game-based model checking algorithm can be extended to deal with KMTSs based on the 2-valued semantics in a more natural way than was needed to deal with the 3-valued semantics.

The 2-valued semantics is aimed at proving φ: it preserves only truth from the abstract model to the concrete one. Therefore, the purpose of the game is also to prove φ's satisfaction. As such, the moves of $\exists$loise in configurations with $EX\varphi'$ formulae need to use $\stackrel{\text{must}}{\longrightarrow}$ transitions, since by the semantics definition, existential formulae are interpreted over *must*-paths. Similarly, the moves of $\forall$belard in configurations with $AX\varphi'$ formulae need to use $\stackrel{\text{may}}{\longrightarrow}$ transitions, since universal formulae are interpreted over *may*-paths. The rest of the moves, as well as the winning criteria remain the same, with the following exception. The transition relation $\stackrel{\text{must}}{\longrightarrow}$ is not necessarily

total. Thus, a configuration of the form $(s, EX\varphi')$ may also be a terminal configuration, if s has no outgoing $\xrightarrow{\text{must}}$ transitions. A play that ends in such a configuration is won by $\forall$belard.

Clearly, the relation between the existence of winning strategies and the satisfaction of the formula, as described in Theorem 2.5 for the concrete game, holds for the new game and the 2-valued semantics. This results from the fact that the change in the allowed moves of the players corresponds exactly to the change in the 2-valued interpretation of a formula over a KMTS.

The model checking algorithm, induced by the game consists of two parts: construction of a game-graph based on the rules of the game, and its coloring. Once the moves for the new game are defined, the game-graph is defined as well. Recall that in the 3-valued case, the resulting game-graph had a different structure and thus the coloring algorithm needed to be changed as well. However, in the 2-valued case, the resulting game-graph has the same structure as a concrete game-graph (with the exception of a new type of terminal nodes): Although the abstract model has two types of transitions for each state, when the game-graph is constructed, the edges become uniform. We no longer distinguish between them, since there is only one type in each node. As a result, in terms of the game-graph there is only one type of edges. Thus, the same coloring algorithm can be applied on the (abstract) game-graph in order to check which player has a winning strategy, with the small change that terminal nodes of the form $(s, EX\varphi')$ need to be colored by F. The correctness of the coloring algorithm, as described in Theorem 2.7 for the concrete case, is maintained since the new game has the same properties as a concrete game: the same possible moves from each configuration (with the type of transitions adapted to match the semantics) and the same winning rules. Thus we are guaranteed that the game-graph is colored by the color of the player that has a winning strategy.

Altogether, we get that the resulting coloring function corresponds to the truth value of the formula over the abstract model, under the 2-valued interpretation of a formula over a KMTS. This is formalized by the following theorem.

Theorem A.2 *Let M be a KMTS and φ a CTL formula. Then, for each $n = (s, \varphi_1) \in G_{M\times\varphi}$:*

1. $[(M,s) \stackrel{2}{\models} \varphi_1] = tt$ *iff* $n = (s, \varphi_1)$ *is colored by* T.
2. $[(M,s) \stackrel{2}{\models} \varphi_1] = ff$ *iff* $n = (s, \varphi_1)$ *is colored by* F.

Intuitively speaking, a node marked by (s, $AX\varphi'$) is now colored by T iff all its sons in the game-graph are colored by T (satisfy φ'), where its sons represent all the states to which s has may transitions. In a similar way, a node marked by (s, $EX\varphi'$) is colored by T iff one of its sons is colored by T (satisfies φ'), where this son represents a state to which s has a must transition. Therefore, the coloring corresponds to the

2-valued semantics.

Complexity: Clearly, the running time of the coloring algorithm remains linear with respect to the size of the game-graph $G_{M\times\varphi}$. The latter is bounded by the size of the underlying KMTS times the length of the CTL formula, i.e. $O(|M| \cdot |\varphi|)$.

A.1 Application to 3-Valued Model Checking

We have the following correspondence between the 2-valued semantics and the 3-valued semantics.

Theorem A.3 *Let M be a KMTS. Then for every CTL formula φ and for every $s \in S$, we have that:*

1. *if* $[(M,s) \stackrel{2}{\models} \varphi] = tt$ *then* $[(M,s) \stackrel{3}{\models} \varphi] = tt$.
2. *if* $[(M,s) \stackrel{2}{\models} \neg\varphi] = tt$ *then* $[(M,s) \stackrel{3}{\models} \varphi] = ff$.
3. *otherwise* $[(M,s) \stackrel{3}{\models} \varphi] = \bot$.

where $\neg\varphi$ denotes the CTL formula equivalent to $\neg\varphi$, with negations pushed to the literals.

Based on Theorem A.3, given an abstract KMTS M_A such that $M_C \preceq M_A$, one may suggest using two instances of the previously described 2-valued model checking algorithm in order to evaluate the 3-valued truth value of φ over M_A, as follows.

First, evaluate φ over M_A using the 2-valued semantics. The constructed game-graph is referred to as the *satisfaction* graph, since it was built for the purpose of proving the satisfaction of φ. If the result is tt for all the initial states, then we have that $[M_A \stackrel{3}{\models} \varphi] = \text{tt}$ and we can conclude that $[M_C \models \varphi] = \text{tt}$.

Otherwise, evaluate $\neg\varphi$ (with negations pushed to the literals) over M_A using the 2-valued semantics. The constructed game-graph is referred to as the *refutation* graph, since it was built for the purpose of proving satisfaction of the negation of φ (which is equivalent to proving refutation of φ). If the result is tt for at least one initial state, we have that $[M_A \stackrel{3}{\models} \varphi] = \text{ff}$ and we can conclude that $[M_C \models \varphi] = \text{ff}$. In addition, a concrete annotated counterexample may be produced from the refutation graph.

This can be better understood using the following observation. Note, that instead of evaluating $\neg\varphi$ using the previous 2-valued game-based model checking algorithm, it is possible to define a game with different rules that is designed to refute the

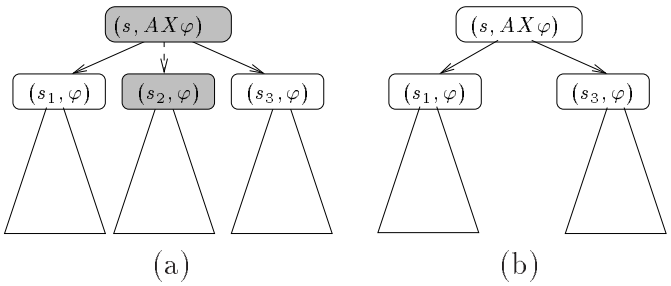

Figure A.1: A satisfaction graph (a) versus a refutation graph (b) for $AX\varphi$

formula φ. In such a game the players use the opposite type of transitions in each configuration (node): $\forall$belard uses must transitions in AX-nodes and $\exists$loise uses may-transitions in EX-nodes. As a result, F is preserved from the corresponding abstract game-graph to the concrete one, but T is not. Note, that the game-graph obtained by these rules is isomorphic to the refutation graph and the result of its coloring is equivalent to the result of the previous algorithm applied on $\neg\varphi$. Obviously, if an initial node in such a game-graph is colored by F, then we can easily find an abstract annotated counterexample by the algorithm `ComputeCounter` described in Chapter 3. The abstract annotated counterexample is guaranteed not to be spurious and can be matched with a concrete one by a greedy algorithm, as described in Chapter 5.

If none of the above holds, we have that $[M_A \stackrel{3}{\models} \varphi] = \bot$. Thus, M_A needs to be refined. One would suggest to try and use both the satisfaction graph and the refutation graph and their coloring functions to find a criterion for refinement. In a sense they complement each other, because they are based on opposite types of transitions. However, these two game-graphs have different nodes (because reachability is also based on opposite transitions), so most chances are that we can not find enough needed information in their intersection. This is demonstrated in Figure A.1, where in the satisfaction graph (a) the initial node $(s, AX\varphi)$ is colored by F since its son (s_2, φ) is colored by F. However, in the refutation graph (b) $(s, AX\varphi)$ is colored by T. Thus, the result of the model checking in indefinite. Unfortunately, the refuting son from the satisfaction graph, (s_2, φ), does not appear in the refutation graph, since it is not a must-son of $(s, AX\varphi)$. Thus, combining the information of both these graphs does not supply enough information for the refinement.

In summary, this approach provides the same information as the 3-valued algorithm about nodes that appear in both the satisfaction and the refutation graphs. Since the *initial* nodes appear in both of them, this approach is sufficient in order to answer the question "$[M \stackrel{3}{\models} \varphi]$?", as accurately as the direct 3-valued approach. However, for the refinement analysis we are interested in the *inner* nodes as well, that are not necessarily mutual to both the graphs. Thus, using two such game-graphs does not provide us with full information (in terms of edges) about all of them. Hence, the 3-valued game-based algorithm is advantageous in terms of the refinement.

Note, that this approach is similar in spirit to the result of translating the KMTS to an equivalent partial Kripke structure (PKS) as described in [22] and then model checking the PKS under the 3-valued semantics by running a standard 2-valued model checker twice, as described in [7].

Appendix B

Memoryless Winning Strategies

Theorem 4.2 describes the correspondence between the existence of winning strategies for the model checking game and the model checking results. Thus, it refers to the existence (or non-existence) of winning strategies for the players. Recall that a strategy for a player is a set of rules telling him (or her) how to proceed from a given configuration. In general, the rules in the strategy can depend on the entire sequence of configurations in the current play that led to the given configuration. Yet, one may also be interested in referring to *memoryless* (or *history-free*) winning strategies, where every rule can depend only on the current configuration of the play and cannot depend on the course of the play up until this configuration (i.e., it should be independent of the prefix of the play). More formally:

Definition B.1 *A* strategy σ for a player P *is a function assigning to every finite sequence of configurations $\vec{C}$ ending in a configuration C, which is in the responsibility of the player P, a configuration C', such that the move from C to C' is a legal move for P. A strategy is* memoryless *iff $\sigma(\vec{C}) = \sigma(\vec{C}')$ whenever $\vec{C}$ and $\vec{C}'$ end in the same configuration.*

The winning strategies described in the proof of Theorem 4.2 are not memoryless. The reason for this is that the paths that are used to construct the winning strategies may contain repetitions of states. The result is that it is possible that the strategy contains different rules for the same configuration (s', φ') based on the position of the state s' in the path, or in other words, based on the position of the configuration in the play. This problem is avoided when using *simple* paths.

Definition B.2

- *An* infinite *path π is said to be* simple *if π is of the form $x \cdot y^\omega$, where $x = s_0, \ldots, s_k$ and $y = s_{k+1}, \ldots, s_n$ such that for every $0 \leq i, j \leq n$: $i \neq j \implies s_i \neq s_j$.*
- *A* finite *path π is said to be* simple *if π is of the form $s_0, \ldots, s_n$ such that for every $0 \leq i, j \leq n$: $i \neq j \implies s_i \neq s_j$.*

The following lemma implies that every non-simple path that is used in the proof of Theorem 4.2 for the construction of a strategy for any of the players can be replaced by a simple one.

Lemma B.3 *Let ψ be a (path) formula of the form $\varphi_1 U \varphi_2$ or $\varphi_1 V \varphi_2$, where φ_1 and φ_2 are CTL formulae. If there exists a (must or may) path π such that $[\pi \stackrel{3}{\models} \psi] = v$ for $v \in \{tt, ff, \bot\}$, then there exists a* simple *path π' of the same type (must or may) such that $[\pi' \stackrel{3}{\models} \psi] = v$ as well.*

Based on Lemma B.3, we conclude that the winning strategies in the proof of Theorem 4.2 can be made memoryless. As a result we get that Theorem 4.2 can be rephrased in terms of *memoryless* winning strategies.

References

[1] A. Asteroth, C. Baier, and U. Assmann. Model checking with formula-dependent abstract models. In *Computer-Aided Verification (CAV)*, volume 2102 of *LNCS*, pages 155–168, 2001.

[2] F. Balarin and A. Sangiovanni-Vincentelli. An itertaive approach to language containment. In *Computer-Aided Verification*, pages 29–40, 1993.

[3] S. Barner, D. Geist, and A. Gringauze. Symbolic localization reduction with reconstruction layering and backtracking. In *Proc. of Conference on Computer-Aided Verification (CAV)*, Copenhagen, Denmark, July 2002.

[4] O. Bernholtz, M. Vardi, and P.Wolper. An automata-theoretic approach to branching-time model cheching. In *Proceedings of the 6th International Conference on Computer Adided Verification (CAV'94)*, volume 818 of *LNCS*, pages 142–155. Springer-Verlag, 1994.

[5] B. Bollig, M. Leucker, and M. Weber. Local parallel model checking for the alternation-free mu-calculus. In *Proceedings of the 9th International SPIN Workshop on Model checking of Software (SPIN '02)*. Springer-Verlag Inc., 2002.

[6] G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *Computer Aided Verification*, pages 274–287, 1999.

[7] G. Bruns and P. Godefroid. Generalized model checking: Reasoning about partial state spaces. *Lecture Notes in Computer Science*, 1877:168–182, 2000.

[8] P. Chauhan, E. Clarke, J. Kukula, S. Sapra, H. Veith, and D.Wang. Automated abstraction refinement for model checking large state spaces using sat based conflict analysis. In *Formal Methods in Computer Aided Design (FMCAD)*, November 2002.

[9] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *12th International Conference on Computer Aided Verification (CAV'00)*, LNCS, Chicago, USA, July 2000.

[10] E. Clarke, O. Grumberg, K. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proceedings of the 32nd Design Automation Conference (DAC'95)*. IEEE Computer Society Press, June 1995.

[11] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT press, December 1999.

[12] E. Clarke, A. Gupta, J. Kukula, and O. Strichman. SAT based abstraction-refinement using ILP and machine leraning techniques. In *Proc. of Conference on Computer-Aided Verification (CAV)*, Copenhagen, Denmark, July 2002.

[13] E. Clarke, S. Jha, Y. Lu, and H. Veith. Tree-like counterexamples in model checking. In *Seventeenth Annual IEEE Symposium on Logic In Computer Science (LICS)*, Copenhagen, Denmark, July 2002.

[14] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16, 5:1512–1542, September 1994.

[15] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *popl4*, pages 238–252, Los Angeles, California, 1977.

[16] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(2), March 1997.

[17] S. Das, D. L. Dill, and S. Park. Experience with predicate abstraction. In *Computer Aided Verification*, pages 160–171, 1999.

[18] D.Peled, A.Pnueli, and L.Zuck. From falsification to verification. In *FSTTCS*, volume 2245 of *LNCS*. Springer-Verlag, 2001.

[19] X. Du, S. A. Smolka, and R. Cleaveland. Local model checking and protocol analysis. *International Journal on Software Tools for Technology Transfer*, 2(3):219–241, 1999.

[20] P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based model checking using modal transition systems. In *Proceedings of CONCUR'01*, 2001.

[21] P. Godefroid and R. Jagadeesan. Automatic abstraction using generalized model checking. In *Proc. of Conference on Computer-Aided Verification (CAV)*, volume 2404 of *LNCS*, pages 137–150, Copenhagen, Denmark, July 2002. Springer-Verlag.

[22] P. Godefroid and R. Jagadeesan. On the expressiveness of 3-valued models. In *Proceedings of VMCAI'2003 (4th Conference on Verification, Model Checking and Abstract Interpretation)*, volume 2575 of *Lecture Notes in Computer Science*, pages 206–222, New York, January 2003. Springer-Verlag.

[23] S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *Proc. of Conference on Computer-Aided Verification (CAV)*, volume 1254 of *LNCS*, pages 72–83. Springer-Verlag, June 1997.

[24] A. Gurfinkel and M. Chechik. Proof-like counter-examples. In *Proceedings of TACAS'03*, April 2003.

[25] T. A. Henzinger, R. Jhala, R. Majumdar, G. C. Necula, G. Sutre, and W. Weimer. Temporal-safety proofs for systems code. In *Proc. of Conference on Computer-Aided Verification (CAV)*, Copenhagen, Denmark, July 2002.

[26] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL)*, pages 58–70, Portland, Oregon, 2002. ACM Press.

[27] M. Huth. Model checking modal transition systems using kripke structures. In *Proceedings of the Third International Workshop on Verification, Model Checking and Abstract Interpretation (VMCAI'02),*, volume 2294 of *LNCS*, pages 302–316, Venice, January 2002. Springer-Verlag.

[28] M. Huth, R. Jagadeesan, and D. Schmidt. Modal transition systems: A foundation for three-valued program analysis. *Lecture Notes in Computer Science*, 2028:155–169, 2001.

[29] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM (JACM)*, 47(2):312–360, 2000.

[30] R. Kurshan. *Computer-Aided-Verification of Coordinating Processes.* Princeton University Press, 1994.

[31] M. Lange. A game based approach to CTL* model checking. In *Proc. summer school MOVEP'2k*, Nantes, France, June 2000.

[32] M. Lange and C. Stirling. Model checking games for CTL*. In *Proc. Conf. on Temporal Logic, ICTL'00*, pages 115–125, Leipzig, Germany, Oct. 2000.

[33] K. Larsen and B. Thomsen. A modal process logic. In *Proceedings of Third Annual Symposium on Logic in Computer Science (LICS)*, pages 203–210. IEEE Computer Society Press, July 1988.

[34] K. G. Larsen. Modal specifications. In J. Sifakis, editor, *Proceedings of the 1989 International Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, France*, volume 407 of *Lecture Notes in Computer Science*. Springer-Verlag, June 1989.

[35] W. Lee, A. Pardo, J.-Y. Jang, G. D. Hachtel, and F. Somenzi. Tearing based automatic abstraction for CTL model checking. In *ICCAD*, pages 76–81, 1996.

[36] M. Leucker. Model checking games for the alternation free mu-calculus and alternating automata. In *6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, September 1999.

[37] J. Lind-Nielsen and H. R. Andersen. Stepwise CTL model checking of state/event systems. In *Computer Aided Verification*, pages 316–327, 1999.

[38] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:11–45, 1995.

[39] K. S. Namjoshi. Certifying model checkers. In *13th Conference on Computer Aided Verification (CAV)*, volume 2102 of *LNCS*. Springer-Verlag, 2001.

[40] K. S. Namjoshi and R. P. Kurshan. Syntactic program transformations for automatic abstraction. In *Proc. of Conference on Computer-Aided Verification (CAV)*, volume 1855 of *LNCS*, pages 435–449, Chicago, IL, USA, July 2000. Springer.

[41] A. Pardo and G. D. Hachtel. Automatic abstraction techniques for propositional mu-calculus model checking. In *Computer Aided Verification*, pages 12–23, 1997.

[42] A. Pardo and G. D. Hachtel. Incremental CTL model checking using BDD subsetting. In *Design Automation Conference*, pages 457–462, 1998.

[43] D. Peled and L. Zuck. From model checking to a temporal proof. In *Proceedings of the 8th international SPIN workshop on Model checking of software*, pages 1–14, Toronto, Ontario, Canada, 2001. Springer-Verlag New York, Inc.

[44] H. Saidi. Model checking guided abstraction and analysis. In *Proceedings of the 7th International Static Analysis Symposium (SAS2000)*, Santa Barbara, CA, Jul 2000.

[45] H. Saidi and N. Shankar. Abstract and model check while you prove. In *Proceedings of the eleventh International Conference on Computer-Aided Verification (CAV99)*, Trento, Italy, Jul 1999.

[46] C. Stirling. Local model checking games. In *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95)*, volume 962 of *LNCS*, pages 1–11, Berlin, Germany, August 1995. Springer.

[47] C. Stirling. *Modal and Temporal Properties of Processes*. Springer, 2001.

[48] L. Tan and R. Cleaveland. Evidence-based model checking. In *14th Conference on Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 455–470, Copenhagen, Denmark, July 2002. Springer Verlag.

[49] E. Yahav, T. Reps, and M. Sagiv. LTL model checking for systems with unbounded number of dynamically created threads and objects. Technical Report TR-1424, Computer Sciences Department, University of Wisconsin, Madison, WI, Mar. 2001.

סכמה מבוססת משחקים לדוגמאות נגדיות ולאלגוריתמי אבסטרקציה-עידון עבור CTL

שרון שוהם

סכמה מבוססת משחקים לדוגמאות נגדיות ולאלגוריתמי אבסטרקציה-עידון עבור CTL

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת תואר
מגיסטר למדעים במדעי המחשב

שרון שוהם

הוגש לסנט הטכניון — מכון טכנולוגי לישראל

כסלו תשס"ד חיפה נובמבר 2003

המחקר נעשה בהנחיית ארנה גרימברג בפקולטה למדעי המחשב

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי

תוכן עניינים

תוכן עניינים (המשך)

רשימת איורים

תקציר מורחב

עבודה זו מנצלת ומרחיבה את הסכמה מבוססת המשחקים של בדיקת מודל CTL [74] לדוגמאות נגדיות ולאלגוריתם אבסטרקציה-עידון הדרגתי.

מטרתנו הראשונה בעבודה זו היא להציע אלגוריתם בדיקת מודל חדש ללוגיקה CTL [11] ביחס למודלים אבסטרקטיים. בדיקת מודל היא גישה מצליחה לבדיקה האם מודל M של מערכת נתונה מספק מפרט הנתון כנוסחה בלוגיקה טמפורלית φ. לחילופין, בדיקת המודל מחשבת את ערך האמת של הנוסחה φ במודל M, כאשר ערך אמת $true$ משמעותו שהנוסחה מסתפקת במודל, וערך אמת $false$ משמעותו שהנוסחה מופרכת במודל. אולם, מודלים קונקרטיים (רגילים) של מערכות אמיתיות נוטים להיות גדולים מאוד. מרחב המצבים שלהם הוא אקספוננציאלי במספר המשתנים במערכת, דבר המוביל לבעיית התפוצצות המצבים. כתוצאה מכך, מתעורר הצורך באבסטרקציה. אבסטרקציה מסתירה חלק מפרטי המערכת, כך שהמודל המתקבל הוא קטן יותר. אבסטרקציות בדרך כלל מתוכננות כך שתהיינה שמרניות ביחס ללוגיקה נתונה. המשמעות היא שאם מודל אבסטרקטי מספק נוסחה בלוגיקה הנתונה, אזי גם המודל הקונקרטי מספק אותה. לעומת זאת, אם המודל האבסטרקטי אינו מספק את הנוסחה, אזי אי אפשר להסיק דבר על המודל הקונקרטי.

ישנן שתי סמנטיקות אפשריות לפירוש נוסחאות בלוגיקה CTL מעל מודלים אבסטרקטיים. הסמנטיקה הדו-ערכית מגדירה נוסחה φ להיות בעלת ערך אמת $true$ או $false$ במודל אבסטרקטי. כאשר ערך האמת הוא $true$, מובטח שהוא ישמר גם במודל הקונקרטי. אולם, ערך אמת $false$ עלול להיות מדומה. הסמנטיקה התלת-ערכית [22] מוסיפה ערך אמת חדש: תחת הסמנטיקה התלת-ערכית יתכן כי ערך האמת של נוסחה במודל אבסטרקטי יהיה לא מוחלט. במקרה זה, לא ניתן כל מידע על ערך האמת במודל הקונקרטי. לעומת זאת, הן ערך האמת $true$ המעיד על הסתפקות, והן ערך האמת $false$ המעיד על הפרכה, נשמרים גם במודל הקונקרטי, כאשר מדובר על הסמנטיקה התלת-ערכית. במילים אחרות, בעוד שאבסטרקציות מעל הסמנטיקה הדו-ערכית הן שמרניות ביחס לתוצאות חיוביות (הסתפקות) בלבד, הרי שאבסטרקציות מעל הסמנטיקה התלת-ערכית הן שמרניות ביחס לתוצאות חיוביות (הסתפקות) ושליליות (הפרכה) גם יחד. כתוצאה מכך, אבסטרקציות מעל הסמנטיקה התלת-ערכית מספקות תוצאות מדויקות לעיתים קרובות יותר הן עבור אימות והן עבור הפרכה.

בעקבות האבחנה הקודמת, אנו מגדירים אלגוריתם בדיקת מודל מבוסס משחקים למודלים אבסטרקטיים ביחס לסמנטיקה התלת-ערכית, שבה מודל אבסטרקטי יכול לשמש הן לאימות והן להפרכה. אולם, עתה מתווספת אפשרות שלישית: בדיקת המודל עשויה להסתיים עם תשובה לא מוחלטת. זהו סימן לכך שהאבסטרקציה הנוכחית אינה מספיק מפורטת על מנת לקבוע מה ערך האמת של התכונה הנבדקת במודל הקונקרטי, ולכן עליה לעבור עידון. עידון הוא תהליך הפוך לאבסטרקציה, שבו מוסיפים פרטים למודל האבסטרקטי.

אבסטרקציה-עידון היא סכמה כללית שבה בדיקת התכונה במודל הנתון מתבצעת באופן איטרטיבי. כל איטרציה מורכבת מבניית מודל אבסטרקטי, ביצוע בדיקת מודל של התכונה במודל האבסטרקטי, ועידון המודל האבסטרקטי במידה ותוצאת בדיקת המודל האבסטרקטי לא מאפשרת להסיק מהי התוצאה במודל הקונקרטי. התהליך נמשך עד לקבלת תוצאה מוחלטת לגבי המודל הקונקרטי. סכמת האבסטרקציה-עידון המסורתית [03, 9] מתאימה לאבסטרקציות דו-ערכיות, שבהן אם התוצאה היא $false$, אזי היא עשויה להיות מדומה. על כן, מטרתו של העידון היא למנוע קבלת תוצאות $false$. תוצאות כאלה מלוות פעמים רבות בדוגמאות נגדיות, לכן במרבית המקרים העידון מבוסס על ניתוח דוגמא נגדית שנמצאה בתהליך בדיקת המודל. בניגוד לגישה הזו, מטרתו של העידון בעבודה שלנו היא למנוע קבלת תוצאות לא מוחלטות, ולשנות תוצאות כאלה ל- $true$ מוחלט או ל- $false$ מוחלט.

אחד היתרונות של עבודה זו טמון בכך שהעידון מופעל רק על החלק של המודל שלגביו התקבלו תוצאות לא מוחלטות. כתוצאה מכך המודל האבסטרקטי לא גדל שלא לצורך. יתר על כן, בדיקת המודל של המודל המעודן נעזרת בתוצאות מוחלטות שהתקבלו באיטרציות קודמות, דבר המוביל לבדיקת מודל הדרגתית. תהליך האבסטרקציה-עידון שאנו מציעים הוא שלם במובן שעבור מודלים קונקרטיים סופיים מובטח שהוא יסתיים עם תשובה מוחלטת.

המטרה הבאה של עבודה זו הינה להשתמש בסכמה מבוססת המשחקים של בדיקת מודל על מנת לספק דוגמאות נגדיות ללוגיקת הזמן המתפצל, CTL. בעת ביצוע בדיקת מודל של מודל M ביחס לתכונה φ, אם המודל אינו מספק את התכונה, אזי כלי בדיקת המודל מנסה לספק דוגמא נגדית. בדרך כלל, דוגמא נגדית היא חלק מהמודל שמסביר מדוע הנוסחה מופרכת במודל. מציאת דוגמאות נגדיות היא תכונה חשובה של בדיקת מודל, אשר מסייעת רבות בגילוי ותיקון שגיאות במערכת הנבדקת.

מרבית כלי בדיקת המודל הקיימים מחזירים בתור דוגמא נגדית מסלול סופי (לצורך הפרכת נוסחאות מהצורה AGp) או מסלול סופי מלווה בחוג (לצורך הפרכת נוסחאות מהצורה AFp[1]) [01, 11]. לאחרונה, גישה זו הורחבה על מנת לספק דוגמאות נגדיות לכל הנוסחאות בלוגיקה ACTL, שהיא החלק האוניברסלי של הלוגיקה CTL [31]. במקרה זה, החלק של המודל שמוחזר בתור דוגמא נגדית הוא בצורת עץ. ישנן עבודות נוספות שעוסקות בהפקת מידע מתהליך בדיקת המודל [34, 81, 93, 84], אך הן מציגות את המידע בצורת הוכחה טמפורלית ולא כחלק מהמודל.

בעבודה זו אנו מספקים דוגמאות נגדיות ללוגיקה CTL בשלמותה. כאמור, כאשר מדובר ב- ACTL, הדוגמאות הנגדיות הן חלק מהמודל. אולם, כאשר מדובר ב- CTL, יש להתמודד עם תכונות קיומיות, בנוסף לתכונות אוניברסליות. על מנת להראות שנוסחה קיומית מהצורה $E\psi$ היא מופרכת, יש להראות מצב התחלתי במודל שכל המסלולים היוצאים ממנו אינם מספקים את ψ. כתוצאה מכך, המבנה של הדוגמא הנגדית נעשה מורכב יותר.

כאשר הדוגמא הנגדית היא כה מורכבת, המשתמש עלול להיתקל בקשיים בהבנתה אם יהיה עליו להסתפק בתת גרף שמהווה חלק מהמודל. על כן, אנו מצמידים לכל מצב בדוגמא הנגדית תת נוסחה של φ שהינה מופרכת על ידי אותו מצב. הנוסחאות האלה, המופרכות במצבים אליהן הן מוצמדות, מספקות את הסיבה לכך שערך האמת של הנוסחה φ הוא $false$ במצב ההתחלתי. לכן, הדוגמא הנגדית המתקבלת, המכונה דוגמא נגדית מסומנת,

[1] המשמעות של AGp היא "בכל מסלול, בכל מצב לאורך המסלול, p מתקיים", בעוד שהמשמעות של AFp היא "לאורך כל מסלול קיים מצב שמספק p".

מהווה כלי נוח לגילוי שגיאות. אנו מציעים אלגוריתם אשר בונה דוגמאות נגדיות מסומנות ומוכיחים שהדוגמאות שהוא מחשב הן מספיקות ומינימליות. בנוסף, אנו דנים במספר דרכים לשימוש והצגת המידע בפועל.

בדיקת מודל מבוססת משחקים [74] עבור CTL היא מסגרת נוחה במיוחד להשגת מטרותינו. משחק בדיקת המודל הוא משחק בין שני שחקנים: אבלרד ($\forall$belard) המפריך אשר מנסה להראות כי המודל M מפריך את הנוסחה φ (יסומן $M \not\models \varphi$), ואלואיז ($\exists$loise) המוכיחה אשר מטרתה להראות כי המודל מספק את הנוסחה (יסומן $M \models \varphi$). לוח המשחק מורכב מזוגות (s, ψ), כאשר s הוא מצב של המודל ו- ψ היא תת-נוסחה של φ, במשמעות שערך האמת של ψ נבחן במצב s. כל אחד מהשחקנים בתורו מתקדם מצומת כזה לצומת שיעזור בהשגת מטרתו: הפרכת או הוכחת ψ במצב s. כל המשחקונים האפשריים במשחק מתוארים על ידי גרף המשחק, שצמתיו הם האלמנטים של לוח המשחק וקשתותיו הן המהלכים האפשריים של השחקנים. הצמתים ההתחלתיים הם זוגות מהצורה (s_0, φ) כאשר s_0 הוא מצב התחלתי במודל. כללי המשחק הם כאלה שמבטיחים כי לאבלרד יש אסטרטגיה מנצחת (כלומר, הוא יכול לנצח ללא תלות במהלכיה של אלואיז) אם"ם $M \not\models \varphi$. לאלואיז יש אסטרטגיה מנצחת אם"ם $M \models \varphi$.

בדיקת המודל מבוצעת על ידי הפעלת אלגוריתם צביעה [5] על גרף המשחק. צומת מהצורה (s, ψ) נצבע ב- T אם"ם לאלואיז יש אסטרטגיה מנצחת החל ממנו, או לחילופין אם"ם הנוסחה ψ מסתפקת על ידי המצב s. הוא נצבע ב- F אם"ם לאבלרד יש אסטרטגיה מנצחת , או לחילופין אם"ם ψ מופרכת על ידי המצב s. בסיום הצביעה, אם כל הצמתים ההתחלתיים צבועים ב- T, אזי ניתן להסיק כי $M \models \varphi$. אם לפחות צומת התחלתי אחד צבוע ב- F, אזי $M \not\models \varphi$ ונרצה לספק דוגמא נגדית.

בעבודה זו אנו מוסיפים אבסטרקציה לדיון. אנו עוסקים בסמנטיקה התלת-ערכית של נוסחאות CTL. ברצוננו לשמר את תכונת הסמנטיקה התלת-ערכית, על פיה גם תוצאות חיוביות וגם תוצאות שליליות הן מוחלטות במובן שהן תקפות גם במודל הקונקרטי. לשם כך, אנו מאפשרים לכל אחד מהשחקנים להיות בעל שני תפקידים במשחק האבסטרקטי. התפקיד הראשון הוא לנצח ולהפריך או להוכיח את התכונה, בהתאם לשחקן (כמו במשחק הרגיל). התפקיד הנוסף הוא לנסות להכשיל את היריב ולמנוע ממנו להשיג את מטרתו. לכן, אבלרד ינסה להפריך את התכונה או למנוע מאלואיז להוכיח אותה. אלואיז, לעומת זאת, תנסה להוכיח את התכונה או למנוע מאבלרד להפריך אותה. במשחקון בודד יתכן עתה כי אף אחד המשחקנים לא ינצח, כך שהמשחקון יסתיים בתיקו. כמו קודם, כללי המשחק הם כאלה שמבטיחים כי לאבלרד יש אסטרטגיה מנצחת אם"ם ערך האמת של φ במודל M הוא $false$. לאלואיז יש אסטרטגיה מנצחת אם"ם ערך האמת הוא $true$. אולם, עתה יתכן כי לאף אחד מהשחקנים אין אסטרטגיה מנצחת , ובמקרה זה ערך האמת של φ במודל M אינו מוחלט.

על מנת לנצל את המשחק החדש לצורך ביצוע בדיקת מודל של מודלים אבסטרקטיים, אנו מציעים אלגוריתם צביעה חדש המשתמש בשלושה צבעים: $T, F, ?$. הצבע T מתאים לערך האמת $true$, F מתאים לערך האמת $false$ ו- $?$ מתאים לערך האמת הלא מוחלט. אם כל הצמתים ההתחלתיים נצבעים ב- T, אזי ניתן להסיק כי ערך האמת של הנוסחה במודל האבסטרקטי הוא $true$. אם איזשהו צומת התחלתי נצבע ב- F ניתן להסיק כי ערך האמת במודל האבסטרקטי הוא $false$. בשני המקרים התוצאה תקפה גם לגבי המודל הקונקרטי. אולם, אם אף אחת מהאפשרויות האלה לא מתקיימת, אזי לא מתקבלת תוצאה מוחלטת. במקרה זה יש לעדן את המודל האבסטרקטי.

אנו בוחרים קריטריון לעידון על ידי בחינת החלק של גרף המשחק הצבוע ב- $?$. בדרך

כלל, לאחר שנבחר קריטריון לעידון, העידון נעשה על ידי פיצול כל המצבים במודל האבסטרקטי. במילים אחרות, בעוד שההחלטה על הקריטריון לעידון היא לוקלית, העידון עצמו הוא גלובלי. אולם, המבנה של גרף המשחק מאפשר לנו לעדן רק את החלק של המודל שלגביו נתקבלו תוצאות בלתי מוחלטות. בנוסף, הוא מאפשר לנו להשתמש בתוצאות מוחלטות שהתקבלו בעבר. כתוצאה מכך, ריצות קודמות אינן מבוזבזות והמודל האבסטרקטי אינו גדל מעבר לנחוץ.

באשר למטרתנו השנייה, אנו מציעים אלגוריתם אשר בונה דוגמא נגדית מסומנת במידה ובדיקת המודל מסתיימת עם תוצאה שלילית, שמשמעותה שהתכונה φ מופרכת על ידי המודל M. תחילה אנו מטפלים במקרה הפשוט יותר שבו בדיקת המודל מבוצעת על מודל קונקרטי. בניית הדוגמא מתבצעת בעזרת גרף הצביעה הצבוע שהתקבל בסיום בדיקת המודל. הדוגמא מתחילה בצומת התחלתי מתוך גרף המשחק שצבעו הוא F. אם בצומת מסויים הנוסחא היא מהצורה $AX\psi$ או $\psi_1 \wedge \psi_2$ אזי הדוגמא הנגדית כוללת את אחד מבניו של הצומת, אשר נצבע ב- F גם כן. בן זה צריך להיבחר בחכמה. אם הנוסחה היא מהצורה $EX\psi$ או $\psi_1 \vee \psi_2$ אזי הדוגמא הנגדית תכלול את כל הבנים של הצומת (שכולם צבועים ב-F). הדוגמא הנגדית המתקבלת היא תת-מודל מסומן של M, יתכן שעם פרישה מסוימת, והיא מספקת את הסיבה המלאה להפרכת התכונה φ במודל.

לאחר שהגדרנו את מושג הדוגמא הנגדית המסומנת, אנו דנים בבניית דוגמא כזו כאשר בדיקת המודל מתבצעת על מודל אבסטרקטי. כיוון שאנו מתבססים על הסמנטיקה התלת-ערכית, מובטח לנו שהקונקרטיזציה של דוגמא נגדית מסומנת אבסטרקטית לא תכשל. זאת כיוון שהאבסטרקציה התלת-ערכית היא שמרנית גם ביחס לתוצאות שליליות של בדיקת מודל. בהסתמך על תכונה זו, אנו מתארים כיצד ניתן להשתמש בהרחבה של האלגוריתם הקונקרטי על מנת לייצר דוגמא נגדית אבסטרקטית וממנה לייצר דוגמא קונקרטית.