Automated Circular Assume-Guarantee Reasoning

Karam Abd Elkader

Automated Circular Assume-Guarantee Reasoning

Research Thesis

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

Karam Abd Elkader

Submitted to the Senate
of the Technion — Israel Institute of Technology
Adar 5776 Haifa February 2016

This research was carried out under the supervision of Prof. Orna Grumberg and Dr. Sharon Shoham, in the Faculty of Computer Science.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisor Prof. Orna Grumberg for being the best supervisor I could have ever hoped for. Orna was always there for me whenever I needed her. I would like to thank Orna for being a great person, for her endless support, friendship and for the continuous encouragement. Thank you so much for introducing me to the wonderful world of research. Certainly, it has been an honor to be your student.

I would like to thank my co-supervisor Dr. Sharon Shoham for being major part of my research. Without her help and without the endless conversations with her about the challenges in my research, I would not be able to make this work successful as it is. I am very indebted to Sharon for being attentive and supportive. Thank you so much Sharon, you are one of the greatest persons I have ever met.

I would like to thank Dr. Corina Pasareanu for the great help I have received from her during my study, for the examples and the baseline code that she has provided, for the great accommodations when I visited NASA research center in the States and for time she dedicated and invested in my research. I am very fortunate that I have had the chance to work with you.

I would like to thank Prof. Ziyad Hanna for the support and the opportunity he gave me by allowing me to combine between study and work. Ziyad, I am very lucky that our paths crossed back in 2011. I can say with certainty that joining your team was one of the smartest decisions I have made in my life. I owe you so much.

Finally, I would like to thank my great father Slameh, my sweet mother Ahlam, my lovely brother Muhammad and my wonderful fascinating sister Sabaa, for the constant encouragement and the moral support they gave me during my studies.

The generous financial help of the Technion is gratefully acknowledged.

Contents

Abstract

Model checking is a successful approach for verifying hardware and software systems. Despite its success, the technique suffers from the state explosion problem which arises due to the huge state space of real-life systems. The size of the model induces high memory and time requirements that may make model checking not applicable to large systems.

One solution to face the state explosion problem is the use of compositional verification, that aims to decompose the verification of a large system into the more manageable verification of its components. To account for dependencies between the components, *assume-guarantee* reasoning defines rules that break-up the global verification of a system into local verification of individual components, using *assumptions* about the rest of the program.

In recent years, compositional techniques have gained significant successes following a breakthrough in the ability to automate assume-guarantee reasoning. However, automation is still restricted to simple acyclic assume-guarantee rules.

In this work, we focus on automating *circular* assume-guarantee reasoning in which the verification of individual components mutually depends on each other. We use a sound and complete circular assume-guarantee rule and we describe how to automatically build the assumptions needed for using the rule. Our algorithm accumulates *joint* constraints on the assumptions based on (spurious) counterexamples obtained from checking the premises of the rule, and uses a SAT solver to synthesize minimal assumptions that satisfy these constraints. To the best of our knowledge, our work is the first to fully automate circular assume-guarantee reasoning.

We implemented our approach and compared it with an established learning-based method that uses an acyclic rule. In all cases, the assumptions generated for the circular rule were significantly smaller, leading to smaller verification problems. Further, on larger examples, we obtained a significant speedup as well.

List of Figures

Chapter 1

Introduction

This work proposes an *automated*, sound and complete *circular* compositional verification technique to address the most central scalability problem in model checking, namely the state-explosion.

Model checking [CGP99] is a widely accepted technique for automatically checking that software systems conform with given properties. Despite its successes, the technique still suffers from the *state explosion problem*, which refers to the worst-case exponential growth of a program's state space with the number of variables and concurrent components. Compositional techniques have shown promise in addressing this problem, by breaking-up the global verification of a program into local, more manageable, verification of its individual components. The environment for each component, consisting of the other program's components, is replaced by a "small" *assumption*, making each verification task easier. This style of reasoning is often referred to as *Assume-Guarantee* (AG) reasoning [MC81, Pnu85].

Progress has been made on automating compositional reasoning using learning and abstraction-refinement techniques for iterative building of the necessary assumptions [CGP03, PGB+08, BPG08]. Other learning-based approaches for automating assumption generation have been proposed as well, e.g. [CCST05, AMN05, CCF+10, CFC+09].

This work has been done mostly in the context of applying a simple compositional *assume-guarantee* rule, where assumptions and properties are related in an *acyclic* manner. For example, in a two component program, suppose component M_1 guarantees property P under assumption A on its environment. Further suppose that M_2 unconditionally guarantees A. Then it follows that the composition $M_1||M_2$ also satisfies P.

However there is another important category of rules that have not been studied for automation. These rules typically involve circular reasoning and use inductive arguments, over time, formulas to be checked, or both, e.g. [MC81, McM98, McM99a, NT00], which makes automation challenging.

Circular Assume-Guarantee rules have been successful in scaling model checking, and have often been more effective than non-circular rules [McM98, McM99a, McM99b, Rus01]. Further, they could naturally exploit the inherent circular dependency exhibited by the verified systems. However, their applicability has been hindered by the manual effort involved in defining the

assumptions.

In this work we propose a novel *circular* compositional verification technique that is fully *automated*. The technique uses the following assume-guarantee circular rule **CIRC-AG**, for proving that $M_1||M_2 \models P$, based on assumptions g_1 and g_2. The rule CIRC-AG is both *sound* and *complete*. Components, properties and assumptions are Labeled Transition Systems (LTSs).

$$\begin{array}{ll} \text{(Premise 1)} & M_1 \models g_2 \triangleright g_1 \\ \text{(Premise 2)} & M_2 \models g_1 \triangleright g_2 \\ \text{(Premise 3)} & g_1||g_2 \models P \\ \hline & M_1||M_2 \models P \end{array}$$

Similar rules have been studied before [McM99a, NT00, GPQ14].

Premises 1 and 2 of the rule are based on induction over time and have the form $M \models A \triangleright P$, which means that for every trace σ of size k, if σ is in the language of M, and its prefix of size $k-1$ is in the language of A then σ is also in the language of P.

Intuitively, premises 1 and 2 prove, in a *compositional and inductive* manner, that every trace in the language of $M_1||M_2$ is also included in the language of $g_1||g_2$. Premise 3 ensures that every trace in the language of $g_1||g_2$ is also included in the language of P, thus the consequence of the rule is obtained. Completeness of the rule stems from the fact that M_1 and M_2 (restricted to appropriate alphabets) can be used for g_1 and g_2 in a successful application of the rule.

The above explanation implies that in a successful application of CIRC-AG, $g_1||g_2$ over-approximates $M_1||M_2$. This means that g_1 overapproximates the part of M_1 restricted to the "intersection" with M_2. Similarly, g_2 overapproximates the part of M_2 restricted to the "intersection" with M_1. In contrast, for the acyclic rule mentioned earlier, the assumption A has to overapproximate M_2 as a whole. Therefore, CIRC-AG can potentially result in substantially smaller assumptions.

We prove soundness and completeness of our CIRC-AG rule, and then turn to its automation. As a first step we suggest an algorithm for checking statements of the form $M \models A \triangleright P$, this algorithm is needed for checking the first two premises. The third premise is checked by standard language inclusion between LTSs (with possibly different alphabets).

To automate Rule CIRC-AG, we develop the Automated Circular Reasoning (ACR) algorithm. ACR works iteratively, with the goal to automate the assumption generation. It runs two algorithms. Algorithm APPLYAG automatically checks whether given assumptions g_1 and g_2 satisfy the premises of the proof rule. If a counterexample is obtained for at least one of the premises, but it cannot be extended into a counterexample for $M_1||M_2 \models P$, the algorithm produces constraints that determine how the assumptions should be refined in order to avoid the same counterexample in subsequent iterations. The other algorithm, GENASSMP, uses a SAT solver to synthesize assumptions g_1 and g_2 that satisfy all the constraints produced by the former algorithm. These algorithms are repeated until assumptions g_1 and g_2 that are suitable for the proof rule are generated, or until a real counterexample is found. ACR always terminates and returns either "$M_1||M_2 \models P$" or "$M_1||M_2 \not\models P$", in which case it also finds a counterexample: a trace in $M_1||M_2$ which is not in P.

For Rule CIRC-AG to be useful in practice, the assumptions g_1 and g_2 must be as small as possible, and clearly smaller than the components M_1 and M_2 they represent. To achieve this, we prove that, at every iteration the generated set of constraints C is the weakest possible in the sense that every (g_1, g_2) which satisfy the rule also satisfy C. Thus, we have no loss of optional assumptions. In addition, the synthesizing algorithm GENASSMP works iteratively with increasing bounds on the total number of states in g_1 and g_2 (i.e., $|g_1| + |g_2|$). Only if the set of constraints is not satisfiable for a given bound, the bound is increased. Thus, g_1 and g_2 are guaranteed to be the smallest (in total number of states) assumptions that satisfy the premises of the rule. Indeed, our experimental results confirm the usefulness of this approach: In all examples $|g_1| + |g_2|$ is smaller than the size of the single assumption produced by an established learning-based method based on an acyclic rule.

Generating assumptions for the circular rule poses unusual challenges. This is because the two assumptions strongly depend on each other and should be generated in a tightly related manner. To achieve this, our constraints can express the fact that a certain trace must or must not be included in the language of an assumption g_i. More importantly, we can express boolean combinations of constraints.

To see why this is needed, consider for example a counterexample for premise 1. Such a counterexample consists of a trace σ and a letter a such that σa is in M_1, σ is in g_2 but σa is not in g_1. In order to eliminate this counterexample we need to either remove σ from g_2 or add σa to g_1. This is done by adding the constraint "the trace σ must not be in g_1 **or** the trace σa must be in g_2". The SAT encoding of this constraint makes sure that at least one of its disjuncts is satisfied. Therefor the trace σa will be no longer a counterexample for premise 1 in subsequent iterations.

We implemented our algorithm and compared it with an established learning-based method that uses the acyclic rule ASYM-AG [CGP03]. Our experiments indicate that the assumptions generated using the circular rule can be much smaller, leading to smaller verification problems, both in the number of explored states and the analysis time.

1.1 Related Work

We discuss here some of the most closely related work.

Assume-guarantee reasoning. In the assume-guarantee paradigm a formula is a triple $< A > M < P >$, where M is a component, P is a property, and A is an assumption about M's environment. The formula is true if whenever M is part of a system satisfying A, then the system must also guarantees P. Assume guarantee reasoning can be applied through several rules. The simplest such rule, called ASYM-AG. It checks if a system composed of components M_1 and M_2 satisfies a property P by checking that M_1 under assumption A satisfies P and that any system containing M_2 as a component satisfies A. The rule can be formulated as follows:

Rule ASYM-AG:

$$\begin{array}{ll} \text{(Premise 1)} & < A > M_1 < P > \\ \text{(Premise 2)} & < true > M_2 < A > \\ \hline & M_1 || M_2 \models P \end{array}$$

In this rule, A denotes an assumption about the environment of M_1. For the use of the rule ASYM-AG to be justified the assumption must be more abstracted than M_2 but still reflect M_2's behavior. Several frameworks have been proposed to support this style of reasoning using the rule ASYM-AG. However, their practical impact has been limited because they require non-trivial human input in defining assumptions that are strong enough to eliminate false violations, but that also respect appropriately the remaining system.

Learning Assumptions. Progress has been made on automating assumption generation for the rule ASYM-AG using learning and abstraction refinement techniques. [CGP03] proposed a framework that fully automates assume-guarantee model checking of safety properties for finite LTSs. They use the learning algorithm L* [Ang87], to compute the assumption. [GPB05] extended the framework of [CGP03] to support a set of symmetric assume-guarantee rules that are sound and complete. In both [CGP03] and [GPB05] the learning-based frameworks are guaranteed to terminate, either stating that the property holds for the system or returning a counterexample if the property is violated.

It has been shown in [CK99] that compositional techniques are particularly effective for well-structured systems that have small interfaces between components. The alphabets of the assumption automata in both [CGP03] and [GPB05] include all the actions in the component interface. In a case study presented in [PG06] it has been observed that a smaller alphabet can be sufficient to prove the property. In this case, using smaller alphabet, assume-guarantee reasoning achieved order of magnitude improvement over non-compositional model checking.

Motivated by the success of a smaller alphabet in learning, [GGP07] and [CS07] proposed automatic process of discovering a smaller alphabet that is sufficient to prove the property. Smaller alphabets mean smaller interfaces, which may lead to smaller assumptions, and hence to smaller verification problems. The process in [GGP07] starts with a small subset of the alphabet and refines it by adding actions to it as necessary until the required property is either shown to hold or shown to be violated by the system. Actions to be added are discovered by an analysis of spurious counterexamples obtained from model checking the components. [CS07] proposed minimizing the assumption alphabet by collecting all the spurious counterexamples encountered so far and finding a minimal eliminating alphabet for them. This is done by reducing the problem to Pseudo-Boolean constraints and solving them using SAT engines for linear constraints over boolean variables. To reduce the assumption even further [BPG08] proposed to combine interface alphabet refinement with orthogonal well-known technique, CEGAR (Counterexample Guided Abstraction Refinement) [CGJ$^+$03]. Using CEGAR, assumptions in [BPG08] basically start from small automata and split states iteratively based on spurious counterexamples that result from the abstraction being too coarse.

The frameworks in [GPB05, PGB$^+$08, GGP07, CS07, BPG08] use the L* [Ang87] automata learning algorithm to iteratively compute assumptions in the form of deterministic finite-state

automata. Other learning-based approaches for automating assumption generation have been suggested as well. The work in [CCF$^+$10] considered a symbolic representation of assumptions and models via Boolean functions. Accordingly they used the CDNF [Bsh95, CW12] learning algorithm of Boolean function in order to learn appropriate assumptions. Another learning-based approach proposed in [CFC$^+$09] used an algorithm for learning a minimal separating automaton as an assumption for rule ASYM-AG. This work uses the observation from [GMF08] that on the one hand an assumption A for the rule ASYM-AG includes all traces of M_1, and on the other hand, it is disjoint from all traces of M_2 that violate P. The work in [GMF08] finds the separating automaton using a SAT solver while [CFC$^+$09] finds it by reducing the problem to the minimization problem of incompletely specified finite state machine.

Our search for minimal assumptions using SAT with an increasing bound is inspired by [GMF08]. However there, a single (separating) assumption is generated for the ASYM-AG rule, while we generate two, mutually dependent assumptions for the CIRC-AG rule.

Circular Rules. Compositional proofs regarding systems of many components often involve apparently **circular** arguments. That is, the correctness of component M_1 must be assumed when verifying component M_2, and vice versa. Circular Rules were shown as valuable tools in the verification of real-world systems in a number of case studies [McM98, HQR98, HQR00, Hoa69, TB97].

Several works [McM99a, NT00] have proposed sound compositional rules for systems with many components that require circular reasoning principles in which properties of other components need to be assumed in proving properties of individual components. An example of a circular rule [McM99a] is given below.

Rule CIRC-1:

$$\frac{\begin{array}{ll}\text{(Premise 1)} & M_1 \models g_2 \prec g_1 \\ \text{(Premise 2)} & M_2 \models g_1 \prec g_2\end{array}}{M_1||M_2 \models P}$$

The rule CIRC-1 is not sound if we interpret $\prec$ as logical implication. The apparent circularity in rule CIRC-1 can be resolved by defining $\prec$ with induction over time, in which case the rule becomes sound. However, it has been shown in [NT00] that it is incomplete. [NT00] studied the incompleteness of rule CIRC-1 and indicates that the reason for CIRC-1 incompleteness is the absence of auxiliary assumptions. This work also suggested a generalization of the rule CIRC-1 which is both sound and complete where h_1 and h_2 are auxiliary assumptions:

Rule CIRC-2:

$$\frac{\begin{array}{ll}\text{(Premise 1)} & M_1 \models (h_2 \wedge g_2) \prec ((h_2 \Rightarrow g_1) \wedge h_1) \\ \text{(Premise 2)} & M_2 \models (h_1 \wedge g_1) \prec ((h_1 \Rightarrow g_2) \wedge h_2)\end{array}}{M_1||M_2 \models G(g_1 \wedge g_2)}$$

Interestingly, [NT00] also shows how proofs derived using the rule CIRC-2 can be translated into proofs using the non-circular rule ASYM-AG. This suggests that circular reasoning is redundant, at least in the case of this rule. However, while circularity is avoided, [NT00] shows that the translation of such proofs is still defined with the $\prec$ operator. That is, inductive reasoning is still needed.

Most of the work on circular assume-guarantee reasoning has been concerned with soundness only. Completeness has rarely been an issue, except for work on assume-guarantee proof systems for deductive verification, see [dRdBH$^+$00]. In the context of compositional model checking, [NT00] was the first work to investigate completeness of circular assume-guarantee rules. It showed a number of circular rules to be incomplete and proposed generalizations which ensure completeness, at the expense of introducing auxiliary variables.

In contrast, [Mai03] introduced the terms backward reasoning and forward reasoning. Backward reasoning corresponds to AG rules in which we match the verification goal against the conclusion of a proof rule and from the premises we can infer what subgoals needed to be established. Example of such sound and complete rule can be found in [NT00]. In forward reasoning, we exploit prior knowledge about components that guarantee properties based on other properties to infer that the system guarantees a conjunction of properties. For example the rule presented in [McM99a] is a forward reasoning rule. The terms backward completeness and forward completeness refer to completeness of backward reasoning AG rules and to completeness of forward reasoning AG rules, respectively. The main result in [Mai03] is that forward reasoning AG rules cannot be sound and complete.

The work in [LDD$^+$13] addresses synthesizing automatically circular compositional proofs based on logical abduction. A key difference is that they refer to a decomposition of a sequential program, while we consider concurrent systems.

1.2 Organization

The rest of this thesis is organized as follows. In the next chapter we give the necessary background for model checking of LTSs. In Chapter 3 we define the Inductive Properties and formally establish the soundness and completeness of the circular rule CIRC-AG. In Chapter 4 we show how to check Inductive Properties, present the ACR algorithm that automates the application of rule CIRC-AG and define Membership Constraints that are essential part of the ACR algorithm. In Chapters 5 and 6 we then describe in detail the algorithms ApplyAG and GenAssmp, respectively. These two algorithms are the main building blocks of the ACR algorithm. In Chapter 7 we prove the correctness of the ACR algorithm and show that it produces minimal assumptions. We provide an experimental evaluation of the proposed algorithm in Chapter 8. Finally, we discuss some conclusions and future work in Chapter 9.

Chapter 2

Preliminaries

In this chapter we provide the necessary background for our work. We introduce labeled transition systems together with their associated operators, and also present how properties are expressed and checked in this context.

Labeled transition systems are defined with respect to a set of *observable actions*. In the sequel, let Act be the universal set of observable actions and let τ denote a local action, unobservable to a component's environment.

Definition 2.1. A *Labeled Transition System* (LTS) M is a quadruple $M = (Q, \alpha M, \delta, q_0)$ where:

- Q is a finite set of states.
- $\alpha M \subseteq Act$ is a finite set of observable actions called the alphabet of M.
- $\delta \subseteq Q \times (\alpha M \cup \{\tau\}) \times Q$ is a transition relation.
- $q_0 \in Q$ is the initial state.

An LTS $M = (Q, \alpha M, \delta, q_0)$ is *nondeterministic* if it contains a τ transition or if there exist $(q, a, q'), (q, a, q'') \in \delta$ such that $q' \neq q''$. Otherwise, M is *deterministic* (denoted as DLTS). We write $\delta(q, a) = \perp$ if there is no q' such that $(q, a, q') \in \delta$. For a DLTS, we write $\delta(q, a) = q'$ to denote that $(q, a, q') \in \delta$.

Paths and Traces A *trace* σ is a sequence of observable actions. For a trace σ, We use σ_i to denote the prefix of σ of length i. A *path* in an LTS $M = (Q, \alpha M, \delta, q_0)$ is a sequence $p = q_0, a_0, q_1, a_1 \cdots, a_{n-1}, q_n$ of alternating states and observable or unobservable actions of M, such that for every $k \in \{0, \ldots, n-1\}$ we have $(q_k, a_k, q_{k+1}) \in \delta$. The *trace* of p, denoted $\sigma(p)$ is the sequence $b_0 b_1 \cdots b_l$ of actions along p, obtained by removing from $a_0 \cdots a_{n-1}$ all occurrences of τ. The set of all traces of paths in M is called the *language* of M, denoted $L(M)$. A trace σ is *accepted* by M if $\sigma \in L(M)$. Note that $L(M)$ is prefix-closed and that the empty trace, denoted by ϵ, is accepted by any LTS.

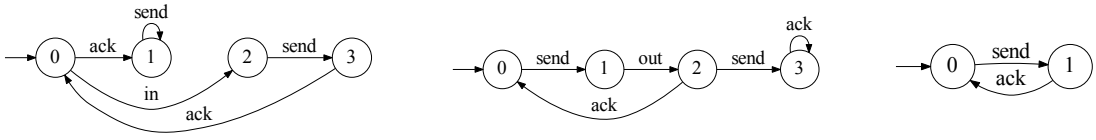


Figure 2.1: LTSs describing the In and Out components and the $Order$ property. $\alpha In = \{in, send, ack\}$, $\alpha Out = \{out, send, ack\}$ and $\alpha Order = \{send, ack\}$.

Note. A non-deterministic LTS can be converted to a deterministic LTS that accepts the same language. However the deterministic LTS might have exponentially more states than the non-deterministic LTS.

Projections For $\Sigma \subseteq Act$, we use $\sigma{\downarrow_\Sigma}$ to denote the trace obtained by removing from σ all occurrences of actions $a \notin \Sigma$. $M{\downarrow_\Sigma}$ is defined to be the LTS over alphabet Σ obtained by renaming to τ all the transitions labeled with actions that are not in Σ. Note that $L(M{\downarrow_\Sigma}) = \{\sigma{\downarrow_\Sigma} \mid \sigma \in L(M)\}$.

Parallel Composition Given two LTSs M_1 and M_2 over alphabet αM_1 and αM_2, respectively, their *interface alphabet* αI consists of their common alphabet. That is, $\alpha I = \alpha M_1 \cap \alpha M_2$. The parallel composition operator $||$ is a commutative and associative operator that combines the behavior of two components by synchronizing on the actions in their interface and interleaving the remaining actions.

Let $M_1 = (Q_1, \alpha M_1, \delta_1, q_{0_1})$ and $M_2 = (Q_2, \alpha M_2, \delta_2, q_{0_2})$ be two LTSs. Then $M_1||M_2$ is an LTS $M = (Q, \alpha M, \delta, q_0)$, where $Q = Q_1 \times Q_2$, $q_0 = (q_{0_1}, q_{0_2})$, $\alpha M = \alpha M_1 \cup \alpha M_2$, and δ is defined as follows where $a \in \alpha M \cup \{\tau\}$:

- if $(q_1, a, q_1') \in \delta_1$ for $a \notin \alpha M_2$, then $((q_1, q_2), a, (q_1', q_2)) \in \delta$ for every $q_2 \in Q_2$,
- if $(q_2, a, q_2') \in \delta_2$ for $a \notin \alpha M_1$, then $((q_1, q_2), a, (q_1, q_2')) \in \delta$ for every $q_1 \in Q_1$, and
- if $(q_1, a, q_1') \in \delta_1$ and $(q_2, a, q_2') \in \delta_2$ for $a \neq \tau$, then $((q_1, q_2), a, (q_1', q_2')) \in \delta$.

Lemma 2.2. *[CGP03] For every $t \in (\alpha M_1 \cup \alpha M_2)^*$, $t \in L(M_1||M_2)$ if and only if $t{\downarrow_{\alpha M_1}} \in L(M_1)$ and $t{\downarrow_{\alpha M_2}} \in L(M_2)$.*

Example 1. Consider the example in Figure 2.1. This is a variation of the example of [CGP03] modified to illustrate circular dependencies. LTSs In and Out have interface alphabet $\{send, ack\}$. Their composition $In||Out$ is an LTS where the transition from state 0 to 1 in component In (labeled with *ack*) never takes place, since there is no corresponding matching transition in component Out. Similarly the transition from state 2 to 3 in component Out (labeled with *send*) never takes place. As a result, $In||Out$ simply repeats the trace $\langle in, send, out, ack \rangle$.

Properties and Satisfiability A *safety property* is defined as an LTS P, whose language $L(P)$ defines the set of acceptable behaviors over the alphabet αP of P. An LTS M over $\alpha M \supseteq \alpha P$ satisfies P, denoted $M \models P$, if $\forall \sigma \in L(M).\sigma{\downarrow_{\alpha P}} \in L(P)$. To check a safety property P, its LTS is transformed into a deterministic LTS, which is also completed by adding an error state π and adding transitions from every state q in the deterministic LTS into π for all the missing outgoing actions of q; the resulting LTS is called an *error LTS*, denoted by P_{err}. Checking that $M \models P$ is done by checking that π is not reachable in $M||P_{err}$.

A trace $\sigma \in \alpha M^*$ is a *counterexample* for $M \models P$ if $\sigma \in L(M)$ but $\sigma{\downarrow_{\alpha P}} \notin L(P)$.

The $Order$ LTS from Figure 2.1 depicts a safety property satisfied by $In||Out$. $Order$ is defined over alphabet $\{send, ack\}$. Note that neither In, nor Out, satisfy this property individually. For example, the trace $\langle in, send, ack, ack\rangle$ of In is a counterexample for $In \models Order$.

Chapter 3

Circular Assume-Guarantee Reasoning

In this chapter we formally establish the soundness and completeness of the circular rule CIRC-AG introduced in Chapter 1. We start by defining inductive properties. CIRC-AG uses formulas of the form $M \models A \triangleright P$, where M is a component, P is a property, and A is an assumption about M's environment. To ensure soundness of the circular rule the assume-guarantee formula is defined using induction over the length of finite traces.

Soundness states that if there exist LTS assumptions g_1 and g_2 that satisfy all premises of CIRC-AG, then $M_1||M_2 \models P$. Completeness states that if $M_1||M_2 \models P$ holds we can always find g_1 and g_2 such that the premises of the rule hold.

3.1 Inductive Properties

Definition 3.1 (The $\triangleright$ operator)**.** Let M, A and P be LTSs over $\alpha M, \alpha A$ and αP respectively, such that $\alpha P \subseteq \alpha M$. We say that $M \models A \triangleright P$ holds if $\forall k \geq 1\ \forall \sigma \in (\alpha M \cup \alpha A)^*$ of length k such that $\sigma{\downarrow_{\alpha M}} \in L(M)$, if $\sigma_{k-1}{\downarrow_{\alpha A}} \in L(A)$ then $\sigma{\downarrow_{\alpha P}} \in L(P)$.

Intuitively, the formula states that if a trace in M satisfies the assumption A up to step $k-1$, it should guarantee P up to step k. As an example consider the LTSs In from Figure 2.1 and g_1 and g_2 from Figure 3.1. Then $In \models g_2 \triangleright g_1$. On the other hand, $In \not\models g_1 \triangleright g_2$ since the trace $\sigma = \langle in, send, ack, ack\rangle \in L(In)$ is such that $\sigma_{k-1}{\downarrow_{\alpha g_1}} = \langle send, ack\rangle \in L(g_1)$, but $\sigma{\downarrow_{\alpha g_2}} = \langle send, ack, ack\rangle \notin L(g_2)$. σ is therefore a counterexample for $In \models g_1 \triangleright g_2$.

Definition 3.2 (Counterexample for $M \models A \triangleright P$)**.** A trace $\sigma \in (\alpha M \cup \alpha A)^*$ of length k is a *counterexample* for $M \models A \triangleright P$ if $\sigma{\downarrow_{\alpha M}} \in L(M)$ and $\sigma_{k-1}{\downarrow_{\alpha A}} \in L(A)$ but $\sigma{\downarrow_{\alpha P}} \notin L(P)$.

3.2 Soundness and Completeness of Rule CIRC-AG

To establish the soundness of rule CIRC-AG we have the following requirements. M_1, M_2 and P are LTSs where $\alpha P \subseteq \alpha M_1 \cup \alpha M_2$. Moreover, g_1, g_2 are LTSs, used as *assumptions* in the

rule, such that $\alpha M_1 \cap \alpha P \subseteq \alpha g_1$ and $\alpha M_2 \cap \alpha P \subseteq \alpha g_2$.

The following lemma is used in the proof of soundness of the rule, but it also provides some insight as to how g_1 and g_2 should be constructed. We will use this insight, and insight from Lemma 3.4, in our algorithm.

Lemma 3.3. *Let g_1 and g_2 be LTS assumptions successfully used in CIRC-AG. Then $M_1||M_2 \models g_1||g_2$.*

Proof. Let g_1 and g_2 be assumptions successfully used in the CIRC-AG rule. Assume, by contradiction, that $M_1||M_2 \not\models g_1||g_2$. Then, let $\sigma \in (\alpha M_1 \cup \alpha M_2)^*$ be a shortest trace such that $\sigma \in L(M_1||M_2)$ but $\sigma{\downarrow_{(\alpha g_1 \cup \alpha g_2)}} \notin L(g_1||g_2)$, i.e. $\sigma{\downarrow_{\alpha g_1}} \notin L(g_1)$ or $\sigma{\downarrow_{\alpha g_2}} \notin L(g_2)$ (by Lemma 2.2). Without loss of generality assume that (1) $\sigma{\downarrow_{\alpha g_1}} \notin L(g_1)$. Note that $\sigma \neq \epsilon$ since $\epsilon \in L(A)$ for every LTS A. In addition $\sigma \in L(M_1||M_2)$ implies in particular that (2) $\sigma{\downarrow_{\alpha M_1}} \in L(M_1)$. Since σ is the shortest trace refuting the relation, $\sigma_{|\sigma|-1}{\downarrow_{(\alpha g1 \cup \alpha g2)}} \in L(g_1||g_2)$, and in particular, (3) $\sigma_{|\sigma|-1}{\downarrow_{\alpha g_2}} \in L(g_2)$. Let σ' be $\sigma{\downarrow_{(\alpha M_1 \cup \alpha g_1 \cup \alpha g_2)}}$. Note that the last letter of σ has to be in $\alpha g_1 \cup \alpha g_2$ (otherwise, $\sigma_{|\sigma|-1}$ is a shorter trace such that $\sigma_{|\sigma|-1} \in L(M_1||M_2)$ but $\sigma_{|\sigma|-1}{\downarrow_{(\alpha g_1 \cup \alpha g_2)}} = \sigma{\downarrow_{(\alpha g_1 \cup \alpha g_2)}} \notin L(g_1||g_2)$). Therefore, σ' is also nonempty. Moreover, σ and σ' share their last action and $\sigma_{|\sigma|-1}{\downarrow_{(\alpha M_1 \cup \alpha g_1 \cup \alpha g_2)}} = \sigma'_{|\sigma'|-1}$. By (2) we get that $\sigma'{\downarrow_{\alpha M_1}}$ is a trace of M_1, whose prefix $\sigma'_{|\sigma'|-1}{\downarrow_{\alpha g_2}} = \sigma_{|\sigma|-1}{\downarrow_{\alpha g_2}}$ is in $L(g_2)$ (by (3)), but by (1), $\sigma'{\downarrow_{\alpha g_1}} = \sigma{\downarrow_{\alpha g_1}}$ is not in $L(g_1)$. This contradicts the fact that $M_1 \models g_2 \triangleright g_1$. □

Note that $M_1||M_2 \models g_1||g_2$ implies that $M_1||M_2 \models g_1$ and $M_1||M_2 \models g_2$. Therefore, Lemma 3.3 states that in a successful application of CIRC-AG, g_i overapproximates the part of M_i restricted to the composition (or if we ignore the different alphabets – intersection) with the other component, as opposed to overapproximating M_i as a whole.

Lemma 3.4. *Let g_1 and g_2 be LTS assumptions successfully used in CIRC-AG, such that $\alpha M_i \cap \alpha P \subseteq \alpha g_i$. Then $M_1||g_2 \models P$ and $M_2||g_1 \models P$.*

Proof. To prove that $M_1||g_2 \models P$, let g_1 and g_2 be LTS assumptions successfully used in CIRC-AG. Assume, by way of contradiction, that $M_1||g_2 \not\models P$. Then, there is a trace $\sigma \in L(M_1||g_2)$ such that $\sigma{\downarrow_{\alpha P}} \notin L(P)$. By Lemma 2.2 we have that (1) $\sigma{\downarrow_{\alpha M_1}} \in L(M_1)$ and $\sigma{\downarrow_{\alpha g_2}} \in L(g_2)$. Since g_2 is an LTS (and hence prefix-closed), $\sigma_{|\sigma|-1}{\downarrow_{\alpha g_2}} \in L(g_2)$ as well. By premise 1, $M_1 \models g_2 \triangleright g_1$. It follows that (2) $\sigma{\downarrow_{\alpha g_1}} \in L(g_1)$. By (1) and (2) and by Lemma 2.2 we get that $\sigma{\downarrow_{\alpha g_1 \cup \alpha g_2}} \in L(g_1||g_2)$. However, since $\sigma{\downarrow_{\alpha P}} \notin L(P)$ (where $\alpha P \subseteq \alpha g_1 \cup \alpha g_2$), we conclude that $g_1||g_2 \not\models P$, in contradiction to premise 3.

The proof of $M_2||g_1 \models P$ is similar to the proof of $M_1||g_2 \models P$. □

The next lemma is used in the completeness proof of CIRC-AG. It shows that under certain restrictions on αg_1 and αg_2, it is possible to verify and falsify properties of the composition of M_1 and M_2 by considering their projections on αg_1 and αg_2. These restrictions guide us in the choice of the alphabets of the assumptions, αg_1 and αg_2, used by our algorithm.

Lemma 3.5. *Let M_1, M_2, P be LTSs over $\alpha M_1, \alpha M_2, \alpha P$ respectively. Let $\alpha g_1 \supseteq \alpha I \cup (\alpha M_1 \cap \alpha P)$ and $\alpha g_2 \supseteq \alpha I \cup (\alpha M_2 \cap \alpha P)$. Then $M_1||M_2 \models P$ if and only if $M_1{\downarrow_{\alpha g_1}}||M_2{\downarrow_{\alpha g_2}} \models P$.* [1]

Proof. ($\Longleftarrow$:) We prove the implication from $M_1{\downarrow_{\alpha g_1}}||M_2{\downarrow_{\alpha g_2}} \models P$ to $M_1||M_2 \models P$. Suppose $M_1{\downarrow_{\alpha g_1}}||M_2{\downarrow_{\alpha g_2}} \models P$. Let $\alpha g_1' = \alpha g_1 \cap \alpha M_1$ and $\alpha g_2' = \alpha g_2 \cap \alpha M_2$. Clearly, $M_1{\downarrow_{\alpha g_1}} = M_1{\downarrow_{\alpha g_1'}}$ and $M_2{\downarrow_{\alpha g_2}} = M_2{\downarrow_{\alpha g_2'}}$. Therefore also $M_1{\downarrow_{\alpha g_1'}}||M_2{\downarrow_{\alpha g_2'}} \models P$. To show that $M_1||M_2 \models P$, consider a trace $\sigma \in L(M_1||M_2)$. We show that $\sigma{\downarrow_{\alpha P}} \in L(P)$.

Let $\sigma' = \sigma{\downarrow_{(\alpha g_1' \cup \alpha g_2')}}$. Then $\sigma' \in L((M_1||M_2){\downarrow_{\alpha g_1' \cup \alpha g_2'}})$. Further, since $\alpha g_1' \subseteq \alpha M_1$ and $\alpha g_2' \subseteq \alpha M_2$, we have that $L((M_1||M_2){\downarrow_{\alpha g_1' \cup \alpha g_2'}}) \subseteq L(M_1{\downarrow_{\alpha g_1'}}||M_2{\downarrow_{\alpha g_2'}})$. Therefore, $\sigma' \in L(M_1{\downarrow_{\alpha g_1'}}||M_2{\downarrow_{\alpha g_2'}})$, hence $\sigma'{\downarrow_{\alpha P}} \in L(P)$. In addition, since $\alpha P \subseteq \alpha g_1' \cup \alpha g_2'$ (since $\alpha g_1' \supseteq \alpha M_1 \cap \alpha P$ and $\alpha g_2' \supseteq \alpha M_2 \cap \alpha P$) it follows that $\sigma{\downarrow_{\alpha P}} = \sigma'{\downarrow_{\alpha P}}$ and therefore $\sigma{\downarrow_{\alpha P}} \in L(P)$.

($\Longrightarrow$:) Suppose $M_1||M_2 \models P$. We show that $M_1{\downarrow_{\alpha g_1}}||M_2{\downarrow_{\alpha g_2}} \models P$. Let $\alpha g_1' = \alpha g_1 \cap \alpha M_1$ and $\alpha g_2' = \alpha g_2 \cap \alpha M_2$. As before, $M_1{\downarrow_{\alpha g_1}} = M_1{\downarrow_{\alpha g_1'}}$ and $M_2{\downarrow_{\alpha g_2}} = M_2{\downarrow_{\alpha g_2'}}$. Therefore it suffices to show that $M_1{\downarrow_{\alpha g_1'}}||M_2{\downarrow_{\alpha g_2'}} \models P$. Consider a trace $\sigma \in L(M_1{\downarrow_{\alpha g_1'}}||M_2{\downarrow_{\alpha g_2'}})$ (i.e. $\sigma \in (\alpha g_1' \cup \alpha g_2')^*$). We show that $\sigma{\downarrow_{\alpha P}} \in L(P)$.

Recall that $\alpha g_1' \subseteq \alpha M_1$ and $\alpha g_2' \subseteq \alpha M_2$. Further, since $\alpha g_1' \cap \alpha M_2 = \alpha I$ and also $\alpha g_2' \cap \alpha M_1 = \alpha I$, we have that $L(M_1{\downarrow_{\alpha g_1'}}||M_2{\downarrow_{\alpha g_2'}}) = L((M_1||M_2){\downarrow_{\alpha g_1' \cup \alpha g_2'}})$. Therefore, $\sigma \in L((M_1||M_2){\downarrow_{\alpha g_1' \cup \alpha g_2'}})$. This means that there exists $\sigma' \in (\alpha M_1 \cup \alpha M_2)^*$ such that $\sigma' \in L(M_1||M_2)$ and $\sigma'{\downarrow_{\alpha g_1' \cup \alpha g_2'}} = \sigma$. Since $M_1||M_2 \models P$, it follows that $\sigma'{\downarrow_{\alpha P}} \in L(P)$. As before, $\alpha P \subseteq \alpha g_1' \cup \alpha g_2'$. Therefore, $\sigma'{\downarrow_{\alpha P}} = \sigma{\downarrow_{\alpha P}}$ and we conclude that $\sigma{\downarrow_{\alpha P}} \in L(P)$. □

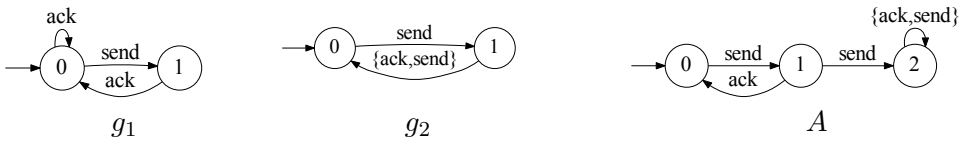
Theorem 3.6. *The Rule CIRC-AG is sound and complete.*

Proof. We start by proving the soundness of the rule CIRC-AG and then turn to proving its completeness:

- **Soundness:** We show that if there exist LTS assumptions g_1 and g_2 that satisfy all premises of CIRC-AG, then $M_1||M_2 \models P$. Assume by way of contradiction that the premises of the CIRC-AG rule hold for some g_1 and g_2 but $M_1||M_2 \not\models P$. Consider a counterexample trace $\sigma \in (\alpha M_1 \cup \alpha M_2)^*$ for $M_1||M_2 \models P$, i.e. σ is a trace of $M_1||M_2$ that violates the property P. In other words $\sigma{\downarrow_{\alpha P}} \notin L(P)$. By Lemma 3.3, we know that $M_1||M_2 \models g_1||g_2$. Therefore, $\sigma{\downarrow_{\alpha g_1 \cup \alpha g_2}} \in L(g_1||g_2)$. In addition, $\alpha P \subseteq \alpha g_1 \cup \alpha g_2$. It follows that $\sigma' = \sigma{\downarrow_{(\alpha g_2 \cup \alpha g_2)}}$ satisfies the following conditions: $\sigma' \in L(g_1||g_2)$ and $\sigma'{\downarrow_{\alpha P}} = \sigma{\downarrow_{\alpha P}} \notin L(P)$. But premise 3 states that such σ' does not exist. Hence we get a contradiction.

- **Completeness:** We show that $M_1||M_2 \models P$ implies that there exist assumptions g_1, g_2 over the alphabets $\alpha g_1 = \alpha M_1 \cap (\alpha M_2 \cup \alpha P)$ and $\alpha g_2 = \alpha M_2 \cap (\alpha M_1 \cup \alpha P)$ respectively

[1] For the implication from $M_1{\downarrow_{\alpha g_1}}||M_2{\downarrow_{\alpha g_2}} \models P$ to $M_1||M_2 \models P$ it suffices to require that $\alpha g_1 \supseteq \alpha M_1 \cap \alpha P$ and $\alpha g_2 \supseteq \alpha M_2 \cap \alpha P$.



g_1 g_2 A

Figure 3.1: LTSs describing the assumptions g_1 and g_2 generated by ACR for verifying $In||Out \models Order$ from Figure 2.1 using the rule CIRC-AG; and LTS describing the assumption A generated with L* for verifying $In||Out \models Order$ from Figure 2.1 using the rule ASYM-AG. $\alpha g_1 = \alpha g_2 = \alpha A = \{send, ack\}$.

that satisfy the premises of the rule. To do so, we consider $g_1 = M_1{\downarrow_{\alpha g_1}}$ and $g_2 = M_2{\downarrow_{\alpha g_2}}$. Clearly premise 1 and 2 are satisfied by these g_1 and g_2 since $M_i \models A \triangleright M_i{\downarrow_{\alpha g_i}}$ for any A. It remains to show that premise 3 holds, i.e. $g_1||g_2 \models P$. Since $M_1||M_2 \models P$ and $\alpha g_i \supseteq \alpha I \cup (\alpha M_i \cap \alpha P)$, the latter follows from Lemma 3.5. □

The completeness proof also shows that for a successful application of the rule it suffices to consider assumptions g_1 and g_2 over $\alpha g_1 = \alpha M_1 \cap (\alpha M_2 \cup \alpha P)$ and $\alpha g_2 = \alpha M_2 \cap (\alpha M_1 \cup \alpha P)$.

Example 2. Consider our running example (Figure 2.1), and consider the assumptions g_1 and g_2 depicted in Figure 3.1, over alphabet $\alpha g_1 = \alpha In \cap (\alpha Out \cup \alpha Order)$ and $\alpha g_2 = \alpha Out \cap (\alpha In \cup \alpha Order)$. In both cases $\alpha g_i = \{send, ack\}$. As stated above, $In \models g_2 \triangleright g_1$. Similarly, $Out \models g_1 \triangleright g_2$. Moreover, $g_1||g_2 \models Order$. It follows that $In||Out \models Order$ can be verified using CIRC-AG with g_1 and g_2 as assumptions.

Chapter 4

Automatic Reasoning with CIRC-AG

In this chapter we describe our ACR iterative algorithm to automate the application of rule CIRC-AG by automating the assumption generation. We also introduce an algorithm for checking inductive properties that is used for checking the first two premises of rule CIRC-AG. Finally, we formally define membership constraints that are used in our algorithm for refining the generated assumptions.

4.1 Checking Inductive Properties

We first introduce a simple algorithm CHECKINDUCTIVEPROPERTY (see Algorithm 4.1) that checks if an inductive property of the form $M \models A \triangleright P$, where $\alpha P \subseteq \alpha M$, holds. If the property does not hold, it returns a counterexample. To do so, we consider the LTS $M||A||P_{err}$. We label its states by (parameterized) propositions err_a, where $a \in \alpha P$. (s_M, s_A, s_P) is labeled by err_a if s_M has an outgoing transition in M labeled by a, but the corresponding transition (labelled by a) leads to π in P_{err}. We then check if a state q labeled by err_a is reachable in $M||A||P_{err}$. If so, then the algorithm returns the trace of a path from q_0 to q extended with action a as a counterexample. Intuitively, such a path to q represents a trace in M that satisfies A (because it is a trace in $M||A$) such that if we extend it by a we get a trace in M violating P.

4.2 ACR Algorithm Overview

In this section we present an iterative algorithm to automate the application of the rule CIRC-AG by automating the assumption generation. Previous work used approximate iterative techniques based on automata learning or abstraction refinement to automate the assumption generation in the context of *acyclic* rules [CGP03, PGB+08, BPG08, CCST05, AMN05, CCF+10, CFC+09]. A different approach [GMF08] used a SAT solver over a set of constraints encoding how the assumptions should be updated to find minimal assumptions; the method was shown to work well in practice, in the context of the same *acyclic* rule. We follow the latter approach here and we adapt it to reasoning for *cyclic* rules and checking inductive assume-guarantee properties. As mentioned, this is challenging due to the mutual dependencies between the two assumptions that

Algorithm 4.1 Checking if $M \models A \triangleright P$

1: **procedure** CHECKINDUCTIVEPROPERTY(M, A, P)
2: $L = (Q, \alpha L, \delta, q_0)$
3: $L \leftarrow M||A||P_{err}$
4: **for each** (s_M, s_A, s_P) in Q **do**
5: $err_a(s_M, s_A, s_P) = \begin{cases} true & a \in \alpha P \wedge \delta_M(s_M, a) \neq \perp \wedge \delta_P(s_P, a) = \pi \\ false & \text{otherwise} \end{cases}$
6: **end for**
7: **if** (exist $q \in Q$ and $a \in \alpha P$ s.t. q is reachable from q_0 in L and $err_a(q) = true$) **then**
8: Let p be a path leading from q_0 to q in L
9: **return** "$\sigma(p)a$ is a counterexample for $M \models A \triangleright P$"
10: **else**
11: **return** "$M \models A \triangleright P$"
12: **end if**
13: **end procedure**

we need to generate. We achieve this by constraining the assumptions with boolean combinations of requirements that certain traces must or must not be included in the language of the updated assumptions.

Algorithm 4.2 describes the overall flow of our Automated Circular Reasoning (ACR) algorithm for checking $M_1||M_2 \models P$ using the rule CIRC-AG.

We fix the alphabet over which the assumptions g_1 and g_2 are computed to be $\alpha g_1 = \alpha M_1 \cap (\alpha M_2 \cup \alpha P)$ and $\alpha g_2 = \alpha M_2 \cap (\alpha M_1 \cup \alpha P)$. By the completeness proof of the rule, this suffices.

ACR maintains a set C of *membership* constraints on g_1 and g_2. At each iteration it calls GENASSMP (described in Chapter 6) to synthesize, using a SAT solver, new minimal assumptions g_1 and g_2 that satisfy all the constraints in C. GENASSMP also receives as input a parameter k which provides a lower bound on the total number of states in the assumptions we look for. k has the property that any pair of LTSs whose total number of states is smaller than k does not satisfy the set of constraints C. The algorithm then invokes APPLYAG (described in Chapter 5) to check the three premises of rule CIRC-AG using the obtained assumptions g_1 and g_2. APPLYAG may return a conclusive result: either "$M_1||M_2 \models P$" or "$M_1||M_2 \not\models P$", in which case ACR terminates. If no conclusive result is obtained, it means that g_1 and g_2 do not satisfy the premises of the rule. Further, the counterexamples demonstrating the falsification of the premises are not suitable for concluding $M_1||M_2 \not\models P$, i.e. they are spurious. In this case APPLYAG returns "continue" together with new membership constraints that determine how the assumptions should be refined. The new constraints are added to C. Note that since the set C of constraints is monotonically increasing, any new pair (g_1', g_2') that satisfies it also satisfies previous sets of constraints. The previous set was satisfied by assumptions whose total size is $|g_1| + |g_2|$ but not smaller. Thus, we should start our search for new (g_1', g_2') from $k = |g_1| + |g_2|$ number of states. k is updated accordingly (line 6).

Example 3. The assumptions g_1 and g_2 from Figure 3.1 used to verify $In||Out \models Order$ with

Algorithm 4.2 Main algorithm for automating rule CIRC-AG for checking $M_1||M_2 \models P$

```
1: procedure ACR(M_1, M_2, P)
2:     Initialize: C = ∅, k = 2
3:     repeat
4:         (g_1, g_2) = GENASSMP(C, k)
5:         (C', Result) = APPLYAG(M_1, M_2, P, g_1, g_2)
6:         C = C ∪ C', k = |g_1| + |g_2|
7:     until (Result ≠ "continue")
8:     return Result
9: end procedure
```

CIRC-AG were obtained by ACR in the 7th iteration. The LTS A from Figure 3.1 describes the assumption obtained with the algorithm of [CGP03], which is based on the acyclic rule ASYM-AG and uses L* for assumption generation. Notice that both g_1 and g_2 are smaller than A (and our experiments show that they can be much smaller in practice). The reason is that, after a successful application of CIRC-AG, $g_1||g_2$ overapproximates $M_1||M_2$. This means that each g_i overapproximates the part of M_i restricted to the composition with the other component. For example g_1 does not include the traces leading to state 1 from *In* since they do not participate in the composition. Similarly g_2 does not include the traces leading to state 3 in *Out*. In contrast, for the acyclic rule, the assumption A has to overapproximate M_2 (*Out*) as a whole. Therefore, CIRC-AG can result in substantially smaller assumptions, as also demonstrated by our experiments.

4.3 Membership Constraints

Membership constraints are used by our algorithm to gather information about traces that need to be in $L(g_i)$ or must not be in $L(g_i)$, for $i = 1, 2$. Thus they allow us to encode dependencies between the languages of the two assumptions $L(g_1)$ and $L(g_2)$. Recall that $\alpha g_1 = \alpha M_1 \cap (\alpha M_2 \cup \alpha P)$ and $\alpha g_2 = \alpha M_2 \cap (\alpha M_1 \cup \alpha P)$. The constraints are defined by formulas with a special syntax and semantics, as defined below.

Definition 4.1 (Syntax)**.** The set of *membership constraints* over $(\alpha g_1, \alpha g_2)$ is defined inductively as follows:

- For $\sigma_1 \in (\alpha g_1)^*$ and $\sigma_2 \in (\alpha g_2)^*$ the following are *atomic* membership constraints: $+(\sigma_1, 1)$, $-(\sigma_1, 1)$, $+(\sigma_2, 2)$, $-(\sigma_2, 2)$.
- if c_1 and c_2 are membership constraints, then $(c_1 \wedge c_2)$ and $(c_1 \vee c_2)$ are membership constraints.

Intuitively $+(\sigma_i, i)$ for i=1,2 constrains $L(g_i)$ to contain σ_i. Similarly $-(\sigma_i, i)$ for i=1,2 constrains $L(g_i)$ not to contain σ_i.

Given a membership constraint formula c, $Strings(c, i)$ is the set of prefixes of all $\sigma \in (\alpha g_i)^*$ such that $+(\sigma, i)$ or $-(\sigma, i)$ is an atomic formula in c.

Definition 4.2 (Semantics)**.** Let c be a membership constraint over $(\alpha g_1, \alpha g_2)$, and let A_1 and A_2 be two LTSs. The satisfaction of c by (A_1, A_2), denoted $(A_1, A_2) \models c$ is defined inductively. $(A_1, A_2) \models c$ if and only if $\alpha A_1 = \alpha g_1$ and $\alpha A_2 = \alpha g_2$, and in addition:

- if c is an atomic formula of the form $+(\sigma, i)$ then $\sigma \in L(A_i)$.
- if c is an atomic formula of the form $-(\sigma, i)$ then $\sigma \notin L(A_i)$.
- if c is of the form $(c_1 \wedge c_2)$ then $(A_1, A_2) \models c_1$ and $(A_1, A_2) \models c_2$.
- if c is of the form $(c_1 \vee c_2)$ then $(A_1, A_2) \models c_1$ or $(A_1, A_2) \models c_2$.

For a set C of membership constraints over $(\alpha g_1, \alpha g_2)$, we say that A_1 and A_2 satisfy C if and only if for every $c \in C$, $(A_1, A_2) \models c$.

For example, a membership constraint of the form $+(\sigma_1, 1) \vee -(\sigma_2, 2)$ requires that $\sigma_1 \in L(g_1)$ or $\sigma_2 \notin L(g_2)$ (or both). As will be shown in section 5.2 our algorithm produces membership constraints formulas over $\alpha g_1 = \alpha M_1 \cap (\alpha M_2 \cup \alpha P)$ and $\alpha g_2 = \alpha M_2 \cap (\alpha M_1 \cup \alpha P)$.

Chapter 5

APPLYAG Algorithm

This chapter is devoted to the description of APPLYAG. Given assumptions g_1,g_2, APPLYAG (see Algorithm 5.1) applies assume-guarantee reasoning by checking the three premises of rule CIRC-AG using g_1 and g_2. In the algorithm we check premises 1, 2, 3 in this order but in fact the order of the checks does not matter and the checks can be done in parallel. If all three premises are satisfied, then, since the rule is sound, it follows that $M_1||M_2 \models P$ holds (and this is returned to the user). Otherwise, at least one of the premises does not hold. Hence a counterexample σ for (at least) one of the premises is found. APPLYAG then checks if the counterexample indicates a real violation for $M_1||M_2 \models P$, as described below. If this is the case, then APPLYAG returns $M_1||M_2 \not\models P$. Otherwise APPLYAG uses the counterexample to compute a set of new membership constraints C and returns "continue" (note that in the first two cases an empty constraint set is returned).

Notation. For readability, in the pseudo-code of APPLYAG (and UPDATECONSTRAINTS) we use $\sigma\downarrow \in L(A)$ and $\sigma\downarrow \notin L(A)$ as a shorthand for $\sigma\downarrow_{\alpha A} \in L(A)$ and $\sigma\downarrow_{\alpha A} \notin L(A)$, respectively.

5.1 Checking Validity of a Counterexample

Given a counterexample σ for one of the premises of the CIRC-AG rule, APPLYAG checks if σ can be extended into a trace in $L(M_1||M_2)$ which does not satisfy P. This check is performed either by APPLYAG directly (if premise 3 fails: in lines 9-16 of APPLYAG) or by algorithm UPDATECONSTRAINTS (if one of the first two premises fails). In essence, a counterexample σ is real if $\sigma\downarrow_{\alpha g_1} \in L(M_1\downarrow_{\alpha g_1})$, $\sigma\downarrow_{\alpha g_2} \in L(M_2\downarrow_{\alpha g_2})$ and $\sigma\downarrow_{\alpha P} \notin L(P)$. This is also stated by the following lemma, which follows from Lemma 2.2 and Lemma 3.5.

Lemma 5.1. *If $\sigma\downarrow_{\alpha g_1} \in L(M_1\downarrow_{\alpha g_1})$, $\sigma\downarrow_{\alpha g_2} \in L(M_2\downarrow_{\alpha g_2})$ and $\sigma\downarrow_{\alpha P} \notin L(P)$, then $M_1||M_2 \not\models P$. Moreover, σ can be extended into a counterexample for $M_1||M_2 \models P$.*

Proof. The first two checks ensure that $\sigma\downarrow_{\alpha g_1 \cup \alpha g_2} \in L(M_1\downarrow_{\alpha g_1}||M_2\downarrow_{\alpha g_2})$, and since $\alpha g_1, \alpha g_2 \supseteq \alpha I$ this ensures that $\sigma\downarrow_{\alpha g_1 \cup \alpha g_2} \in L((M_1||M_2)\downarrow_{\alpha g_1 \cup \alpha g_2})$. Since $\alpha P \subseteq \alpha g_1 \cup \alpha g_2$, the third

check ensures that $(\sigma{\downarrow_{\alpha g_1 \cup \alpha g_2}}){\downarrow_{\alpha P}} = \sigma{\downarrow_{\alpha P}} \notin L(P)$. Therefore $\sigma{\downarrow_{\alpha g_1 \cup \alpha g_2}}$ is a counterexample for $M_1{\downarrow_{\alpha g_1}}||M_2{\downarrow_{\alpha g_2}} \models P$, and by Lemma 3.5, $\sigma{\downarrow_{\alpha g_1 \cup \alpha g_2}}$ can be extended into a counterexample for $M_1||M_2 \models P$. □

For example, in line 10 of Algorithm 5.1, $\sigma \in (\alpha g_1 \cup \alpha g_2)^*$ is a counterexample for premise 3, hence $\sigma{\downarrow_{\alpha P}} \notin L(P)$. It therefore suffices to check if $\sigma{\downarrow_{\alpha g_1}} \in L(M_1{\downarrow_{\alpha g_1}})$ and $\sigma{\downarrow_{\alpha g_2}} \in L(M_2{\downarrow_{\alpha g_2}})$ in order to conclude that a real counterexample exists (line 11). Similarly, in Algorithm 5.2, $\sigma a \in (\alpha M_i \cup \alpha g_j)^*$ is a counterexample for premise i for $i \in \{1, 2\}$, hence $\sigma a{\downarrow_{\alpha M_i}} \in L(M_i)$, and since $\alpha g_i \subseteq \alpha M_i$, also $\sigma a{\downarrow_{\alpha g_i}} \in L(M_i{\downarrow_{\alpha g_i}})$. In line 3, the algorithm then checks if, in addition, $\sigma a{\downarrow_{\alpha g_j}} \in L(M_j{\downarrow_{\alpha g_j}})$ and $\sigma a{\downarrow_{\alpha P}} \notin L(P)$. If these conditions hold then by Lemma 5.1 the counterexample is real (line 5).

5.2 Computation of New Membership Constraints based on Counterexamples

When the counterexample found for one of the premises does not produce a real counterexample for $M_1||M_2 \models P$, then APPLYAG (or UPDATECONSTRAINTS) analyzes the counterexample and computes new membership constraints to *refine* the assumptions. In essence, these constraints encode whether the counterexample trace (or a restriction of it) should be added to or removed from the languages of the two assumptions such that future checks will not produce the same counterexample again.

If premise 3 does not hold, i.e. $g_1||g_2 \not\models P$ and the reported counterexample σ is found not to be real then it should be removed from $L(g_1)$ or from $L(g_2)$ (in this way the trace will no longer be present in the composition $g_1||g_2$ for the assumptions computed in subsequent iterations). Therefore in line 14, APPLYAG adds the corresponding constraint $(-(\sigma{\downarrow_{\alpha g_1}}, 1) \vee -(\sigma{\downarrow_{\alpha g_2}}, 2))$ to C.

If either premise 1 or 2 does not hold, i.e. $M_i \not\models g_j \triangleright g_i$, then the analysis of the counterexample $\sigma_i a_i$ (for i=1 or 2) and the addition of constraints (if needed) are performed by UPDATECONSTRAINTS (see Algorithm 5.2). Specifically, in this case $\sigma_i a_i$ should be added to $L(g_i)$ or its prefix σ_i should be removed from $L(g_j)$ (where $j \neq i$). In both cases, this ensures that checking $M_i \not\models g_j \triangleright g_i$ in subsequent iterations will no longer produce the same counterexample (see Definition 3.1).

We add this constraint in line 18 of Algorithm 5.2, where C is updated with $(-(\sigma{\downarrow_{\alpha g_j}}, j) \vee +(\sigma a{\downarrow_{\alpha g_i}}, i)$. Although this simple refinement would work for all cases, note that Algorithm 5.2, uses a more involved refinement. The reason is that we exploit the properties stated in Lemma 3.3 and Lemma 3.4, to detect more elaborate constraints; using the lemma and analyzing both σ and σa allows us to *accelerate* the refinement process.

For example, in line 25, the subconstraint $+(\sigma a{\downarrow_{\alpha g_i}}, i)$ is conjoined with $-(\sigma a{\downarrow_{\alpha g_j}}, j)$. This is because Lemma 3.3 establishes that $M_i||g_j \models P$ is a necessary condition for a successful application of CIRC-AG. Therefore since $\sigma a{\downarrow_{\alpha g_i}} \in L(M_i{\downarrow_{\alpha g_i}})$ and $\sigma a{\downarrow_{\alpha P}} \notin L(P)$, then $\sigma a{\downarrow_{\alpha g_j}}$ must not be in $L(g_j)$. Explanations of other cases appear as comments in the pseudocode.

Algorithm 5.1 Applying CIRC-AG with g_1 and g_2, and constraint updating.

```
1:  procedure APPLYAG(M_1, M_2, P, g_1, g_2)
2:      if M_1 ⊭ g_2 ▷ g_1 then
3:          Let σ_1 a_1 be a counterexample for M_1 ⊭ g_2 ▷ g_1
4:          return UPDATECONSTRAINTS(1, 2, M_1, M_2, P, σ_1 a_1)
5:      else if M_2 ⊭ g_1 ▷ g_2 then
6:          Let σ_2 a_2 be a counterexample for M_2 ⊭ g_1 ▷ g_2
7:          return UPDATECONSTRAINTS(2, 1, M_2, M_1, P, σ_2 a_2)
8:      else if g_1||g_2 ⊭ P then
9:          Let σ be a counterexample for g_1||g_2 ⊭ P
10:         if (σ↓ ∈ L(M_1 ↓ αg_1) && σ ↓∈ L(M_2 ↓ αg_2)) then
11:             return (∅, "M_1||M_2 ⊭ P")
12:         else      // σ ∉ L(M_1↓_{αg_1}||M_2↓_{αg_2}), σ↓ ∉ L(P)
13:             // Remove σ from g_1 or remove σ from g_2
14:             C = {(−(σ↓_{αg_1}, 1) ∨ −(σ↓_{αg_2}, 2))}
15:             return (C, "continue")
16:         end if
17:     else
18:         return (∅, "M_1||M_2 ⊨ P")
19:     end if
20: end procedure
```

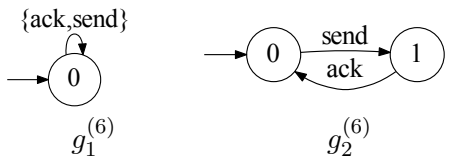


Figure 5.1: LTSs produced in the 6th iteration of ACR.

Example 4. Consider the LTSs from Figure 5.1, produced in the 6th iteration of ACR. When trying to apply CIRC-AG with these assumptions, APPLYAG obtains the trace $\langle send, out, send\rangle$ as a counterexample for $Out \models g_1^{(6)} \triangleright g_2^{(6)}$ (premise 2).

Since $\langle send, out, send\rangle\downarrow_{\alpha g_1} \notin L(In\downarrow_{\alpha g_1})$, the counterexample turns out to be spurious, and after checking the additional conditions in UPDATECONSTRAINTS, $-(\langle send\rangle, 1) \vee (+(\langle send, send\rangle, 2) \wedge -(\langle send, send\rangle, 1))$ is produced in line 25 as a membership constraint in order to eliminate it in the following iterations.

In the following we state the progress obtained by assumption refinement, based on spurious counterexamples.

Lemma 5.2. *Let σ be a spurious counterexample obtained for premise $i \in \{1, 2, 3\}$ of CIRC-AG with respect to assumptions g_1, g_2 and let C be the updated set of constraints. Then any pair of LTSs g_1' and g_2' such that $(g_1', g_2') \models C$ will no longer exhibit σ as a counterexample for premise i of CIRC-AG.*

Algorithm 5.2 Computation of constraints based on a counterexample for $M_i \models g_j \triangleright g_i$.

1: // σa is a counterexample for $M_i \models g_j \triangleright g_i$, i.e. $\sigma a{\downarrow} \in L(M_i), \sigma{\downarrow} \in L(g_j), \sigma a{\downarrow} \notin L(g_i)$
2: **procedure** UPDATECONSTRAINTS($i, j, M_i, M_j, P, \sigma a$)
3: **if** $\sigma a{\downarrow} \in L(M_j{\downarrow}_{\alpha g_j})$ and $\sigma a{\downarrow} \notin L(P)$ **then**
4: // $\sigma a{\downarrow} \in L(M_i{\downarrow}_{\alpha g_i} || M_j{\downarrow}_{\alpha g_j})$ and $\sigma a{\downarrow} \notin L(P)$
5: **return** ($\emptyset$, "$M_i || M_j \not\models P$")
6: **if** $\sigma a{\downarrow} \in L(M_j{\downarrow}_{\alpha g_j})$ and $\sigma a{\downarrow} \in L(P)$ **then**
7: // Add σa to both g_i and g_j to ensure $M_1{\downarrow}_{\alpha g_1} || M_2{\downarrow}_{\alpha g_2} \models g_1 || g_2$ (Lemma 3.3)
8: $C = \{+(\sigma a{\downarrow}_{\alpha g_i}, i), +(\sigma a{\downarrow}_{\alpha g_j}, j)\}$
9: **if** $\sigma a{\downarrow} \notin L(M_j{\downarrow}_{\alpha g_j})$ and $\sigma{\downarrow} \in L(M_j{\downarrow}_{\alpha g_j})$ and $\sigma{\downarrow} \notin L(P)$ **then**
10: // $\sigma{\downarrow} \in L(M_i{\downarrow}_{\alpha g_i} || M_j{\downarrow}_{\alpha g_j})$ and $\sigma{\downarrow} \notin L(P)$
11: **return** ($\emptyset$, $M_i || M_j \not\models P$)
12: **if** $\sigma a{\downarrow} \notin L(M_j{\downarrow}_{\alpha g_j})$ and $\sigma{\downarrow} \in L(M_j{\downarrow}_{\alpha g_j})$ and $\sigma{\downarrow} \in L(P)$ **then**
13: // $\sigma \in L(M_1{\downarrow}_{\alpha g_1} || M_2{\downarrow}_{\alpha g_2})$, thus σ cannot be removed from g_j (Lemma 3.3)
14: // Add σa to g_i.
15: $C = \{+(\sigma a{\downarrow}_{\alpha g_i}, i)\}$
16: **if** $\sigma a{\downarrow} \notin L(M_j{\downarrow}_{\alpha g_j})$ and $\sigma{\downarrow} \notin L(M_j{\downarrow}_{\alpha g_j})$ and $\sigma a{\downarrow} \in L(P)$ **then**
17: // Remove σ from g_j or add σa to g_i
18: $C = \{(-(\sigma{\downarrow}_{\alpha g_j}, j) \vee +(\sigma a{\downarrow}_{\alpha g_i}, i)\}$
19: **if** $\sigma a{\downarrow} \notin L(M_j{\downarrow}_{\alpha g_j})$ and $\sigma{\downarrow} \notin L(M_j{\downarrow}_{\alpha g_j})$ and $\sigma a{\downarrow} \notin L(P)$ and $\sigma{\downarrow} \notin L(P)$ **then**
20: // Remove σ from g_j (Because of Lemma 3.4)
21: $C = \{-(\sigma{\downarrow}_{\alpha g_j}, j)\}$
22: **if** $\sigma a{\downarrow} \notin L(M_j{\downarrow}_{\alpha g_j})$ and $\sigma{\downarrow} \notin L(M_j{\downarrow}_{\alpha g_j})$ and $\sigma a{\downarrow} \notin L(P)$ and $\sigma{\downarrow} \in L(P)$ **then**
23: // Remove σ from g_j or (add σa to g_i and remove it from g_j)
24: // In the latter case removal of σa from g_j is due to Lemma 3.4
25: $C = \{(-(\sigma{\downarrow}_{\alpha g_j}, j) \vee (+(\sigma a{\downarrow}_{\alpha g_i}, i) \wedge -(\sigma a{\downarrow}_{\alpha g_j}, j)))\}$
26: **return** (C, "continue")
27: **end procedure**

Proof. Let σ be a spurious counterexample that has been produced by checking premise i of rule CIRC-AG:

- $i \in \{1, 2\}$: When $\sigma = \sigma' a$ is a spurious counterexample for premise i where $i \in \{1, 2\}$ then one of the following constraints is added by Algorithm 5.2 to the updated set of constraints C:
  - c = $+(\sigma' a{\downarrow}_{\alpha g_i}, i) \wedge +(\sigma' a{\downarrow}_{\alpha g_j}, j)$ in Algorithm 5.2, line 8.
  - c = $+(\sigma' a{\downarrow}_{\alpha g_i}, i)$ in Algorithm 5.2, line 15.
  - c= $-(\sigma'{\downarrow}_{\alpha g_j}, j)$ in Algorithm 5.2, line 21.
  - c= $-(\sigma'{\downarrow}_{\alpha g_j}, j) \vee +(\sigma' a{\downarrow}_{\alpha g_i}, i)$ in Algorithm 5.2, line 18.
  - c= $-(\sigma'{\downarrow}_{\alpha g_j}, j) \vee (+(\sigma' a{\downarrow}_{\alpha g_i}, i) \wedge -(\sigma' a{\downarrow}_{\alpha g_j}, j))$ in Algorithm 5.2, line 25.

  Since (g_1', g_2') satisfies C it implies that (g_1', g_2') satisfies c as well. It is easy to see that since c is one of the above constraints, adding it to C guarantees that $\sigma' a$ can no longer

be a counterexample for premise i with the assumptions g_1', g_2'.

- $i = 3$: σ is a spurious counterexample that has been produced from checking premise 3 in CIRC-AG. Thus, the constraint $c = -(\sigma{\downarrow}_{\alpha g_1}, 1) \vee -(\sigma{\downarrow}_{\alpha g_2}, 2)$ is added by Algorithm 5.1 to the updated set of constraints C (Algorithm 5.1, Line 14). Since (g_1', g_2') satisfies C it follows that (g_1', g_2') satisfies c as well. It implies that $\sigma{\downarrow}_{\alpha g_1}$ is not in $L(g_1')$ or $\sigma{\downarrow}_{\alpha g_2}$ is not in $L(g_2')$ therefore, σ can not be a counterexample for premise 3 when applying the CIRC-AG rule with the assumptions (g_1', g_2'). □

Corollary 5.3. *Any pair of LTSs g_1' and g_2' such that $(g_1', g_2') \models C$ is different from every previous pair of LTSs considered by the algorithm.*

The following two lemmas state that the added membership constraints do not over-constrain the assumptions. They ensure that the "desired" assumptions that enable to verify (Lemma 5.4) or falsify (Lemma 5.5) the property are always within reach.

Lemma 5.4. *Suppose $M_1||M_2 \models P$ and let g_1 and g_2 be LTSs that satisfy the premises of rule CIRC-AG. Then (g_1, g_2) satisfy every set of constraints C produced by* APPLYAG.

Proof. To prove the lemma, we need to show that every g_1, g_2 that satisfy the premises of rule CIRC-AG satisfy all the forms of constraints that are being produced by both Algorithm 5.1 and Algorithm 5.2.

Algorithm 5.1: in line 14, the constraint is of the form $c = (-(\sigma{\downarrow}_{\alpha g_1}, 1) \vee -(\sigma{\downarrow}_{\alpha g_2}, 2))$. By reaching line 14, it follows that $\sigma{\downarrow}_{\alpha P}$ is not in $L(P)$. We prove by way of contradiction that $(g_1, g_2) \models c$. Suppose that $(g_1, g_2) \not\models c$ then $\sigma{\downarrow}_{(\alpha g_1 \cup \alpha g_2)}$ is in $L(g_1||g_2)$ but $\sigma{\downarrow}_{\alpha P}$ is not in $L(P)$. Hence, we have a contradiction to the fact that $g_1||g_2 \models P$.

Algorithm 5.2: For every (i, j) in {(1,2),(2,1)} :

- in line 8 we add the following two constraints $+(\sigma a{\downarrow}_{\alpha g_i}, i)$ and $+(\sigma a{\downarrow}_{\alpha g_j}, j)$. When Algorithm 5.2 reaches line 8, we know that $\sigma a{\downarrow} \in L(M_1{\downarrow}_{\alpha g_1}||M_2{\downarrow}_{\alpha g_2})$. Therefore, by Lemma 3.3 we get that $\sigma a{\downarrow} \in L(g_i)$ and $\sigma a{\downarrow} \in L(g_j)$. Hence, it follows that $(g_1, g_2) \models +(\sigma a{\downarrow}_{\alpha g_i}, i)$ and $(g_1, g_2) \models +(\sigma a{\downarrow}_{\alpha g_j}, j)$.
- in line 15 we add the following constraint $+(\sigma a{\downarrow}_{\alpha g_i}, i)$. When Algorithm 5.2 reaches line 15, we know that (1) $\sigma a{\downarrow} \in L(M_i)$, $\sigma{\downarrow} \in L(M_i)$ and $\sigma{\downarrow} \in L(M_j{\downarrow}_{\alpha g_j})$. Assume by way of contradiction that $(g_1, g_2) \not\models +(\sigma a{\downarrow}_{\alpha g_i}, i)$, it follows that $\sigma a{\downarrow} \notin L(g_i)$. By (1) we get that $\sigma a{\downarrow}_{\alpha M_i}$ is a trace of M_i, whose prefix $\sigma{\downarrow}_{\alpha g_j}$ is in $L(g_j)$ (by (1) and Lemma 3.3), but $\sigma a{\downarrow}_{\alpha g_i}$ is not in $L(g_i)$. This contradicts the fact that the premise $M_i \models g_j \prec g_i$ holds.
- in line 18 we add a constraint of the form $c = (-(\sigma{\downarrow}_{\alpha g_j}, j) \vee +(\sigma a{\downarrow}_{\alpha g_i}, i))$. We know that $\sigma a{\downarrow}_{\alpha M_i}$ is a trace of M_i. Assume by way of contradiction that (g_1, g_2) does not satisfy c, then we get that $\sigma{\downarrow}_{\alpha g_j}$ is in $L(g_j)$ and $\sigma a{\downarrow}_{\alpha g_i}$ is not in $L(g_i)$. This contradicts the fact that the premise $M_i \models g_j \triangleright g_i$ holds.

- in line 21 we add a constraint of the form $c = -(\sigma{\downarrow}_{\alpha g_j}, j)$. When Algorithm 5.2 reaches line 21 we know that (1) $\sigma{\downarrow}_{\alpha M_i}$ is in $L(M_i)$ but $\sigma{\downarrow}_{\alpha P}$ is not in $L(P)$. Assume by way of contradiction that (g_1, g_2) does not satisfy c then we get that (2) $\sigma{\downarrow}_{\alpha g_j}$ is in $L(g_j)$. By (1) and (2) we get that σ is in $L(M_i||g_j)$, which contradicts Lemma 3.4 that states $M_i||g_j \models P$.
- in line 25 we add a constraint of the form $c = (-(\sigma{\downarrow}_{\alpha g_j}, j) \vee (+(\sigma a{\downarrow}_{\alpha g_i}, i) \wedge -(\sigma a{\downarrow}_{\alpha g_j}, j)))$. Assume by way of contradiction that (g_1, g_2) does not satisfy c, it follows that (1) $\sigma{\downarrow}_{\alpha g_j}$ is in $L(g_j)$ and $\sigma a{\downarrow}_{\alpha g_i}$ is not in $L(g_i)$ or $\sigma a{\downarrow}_{\alpha g_j}$ is in $L(g_j)$. We know that $\sigma a{\downarrow}_{\alpha M_i}$ is in $L(M_i)$ and that $\sigma a{\downarrow}_{\alpha P}$ is not in $L(P)$. g_1 and g_2 satisfy premise (i) of rule CIRC-AG then by (1) we get the $\sigma a{\downarrow}_{\alpha g_i}$ is in $L(g_i)$, it implies that $\sigma a{\downarrow}_{\alpha g_j}$ is in $L(g_j)$ (Again by 1). Therefore σa is in $L(g_1||g_2)$ and this contradicts the fact the g_1 and g_2 satisfy premise (3) of rule CIRC-AG. □

Lemma 5.5. *Let $g_1 = M_1{\downarrow}_{\alpha g_1}$ and $g_2 = M_2{\downarrow}_{\alpha g_2}$. Then (g_1, g_2) satisfy every set of constraints C produced by* APPLYAG.

Proof. To prove the lemma we need to show that $(M_1{\downarrow}_{\alpha g_1}, M_2{\downarrow}_{\alpha g_2})$ satisfies all the forms of constraints that are being produced by Algorithm 5.1 and Algorithm 5.2.

First note that $M_1{\downarrow}_{\alpha g_1}$ and $M_2{\downarrow}_{\alpha g_2}$ satisfy premises 1 and 2 of CIRC-AG. Furthermore, $M_1||M_2 \models M_1{\downarrow}_{\alpha g_1}||M_2{\downarrow}_{\alpha g_2}$. Therefore, similarly to the proof of Lemma 5.4, we get that $M_1{\downarrow}_{\alpha g_1}$ and $M_2{\downarrow}_{\alpha g_2}$ satisfy all constraints produced in Algorithm 5.2, lines 8, 15 and 18 (These constraints do not require the assumptions to satisfy the third premise of rule CIRC-AG). It remains to show that $M_1{\downarrow}_{\alpha g_1}$ and $M_2{\downarrow}_{\alpha g_2}$ satisfy the constraints that are produced in Algorithm 5.1, line: 14 and Algorithm 5.2, lines: 21, 25.

Algorithm 5.1: in line 14, we add a constraint of the form $c = (-(\sigma{\downarrow}_{\alpha g_1}, 1) \vee -(\sigma{\downarrow}_{\alpha g_2}, 2))$. If $M_1||M_2 \models P$ then by Lemma 5.4 and by Lemma 3.5 we get that $(M_1{\downarrow}_{\alpha g_1}, M_1{\downarrow}_{\alpha g_2})$ satisfies c. Otherwise $M_1||M_2 \not\models P$, suppose that $(M_1{\downarrow}_{\alpha g_1}, M_1{\downarrow}_{\alpha g_2})$ does not satisfy c, it follows that ($\sigma{\downarrow} \in L(M_1{\downarrow}_{\alpha g_1})$ and $\sigma{\downarrow} \in L(M_2{\downarrow}_{\alpha g_2})$). It implies that the condition in line 10 holds and this contradicts the fact that c has been added as a constraint.

Algorithm 5.2: For every (i, j) in {(1,2),(2,1)} :

- in line 21, we add the constraint of the form $c = -(\sigma{\downarrow}_{\alpha g_j}, j)$. Suppose that $(M_1{\downarrow}_{\alpha g_1}, M_2{\downarrow}_{\alpha g_2})$ does not satisfy c, it follows that $\sigma{\downarrow} \in L(M_j{\downarrow}_{\alpha g_j})$, which means that the condition in line 19 does not hold. This contradicts the fact that c has been added as a constraint.
- in line 25, we add a constraint of the form $c = (-(\sigma{\downarrow}_{\alpha g_j}, j) \vee (+(\sigma a{\downarrow}_{\alpha g_i}, i) \wedge -(\sigma a{\downarrow}_{\alpha g_j}, j)))$. Suppose that $(M_1{\downarrow}_{\alpha g_1}, M_2{\downarrow}_{\alpha g_2})$ does not satisfy c then it follows that $\sigma{\downarrow}_{\alpha g_j}$ is in $L(M_j{\downarrow}_{\alpha g_j})$. Thus, the condition in line 22 does not hold and this contradicts the fact that c has been added as a constraint.

□

Chapter 6

GENASSMP Algorithm

In this chapter, we describe how we generate assumptions that satisfy the constraints that we collect in Algorithm 5.1. Given a set of membership constraints C, and a lower bound k on the total number of states in $|g_1| + |g_2|$, GENASSMP (see Algorithm 6.1) computes assumptions g_1 and g_2 that satisfy C. Similarly to previous work [GMF08] we build assumptions as deterministic LTSs (even though APPLYAG is not restricted to deterministic LTSs). Technically, for each value of k starting from the given k, GENASSMP encodes the structural requirements of the desired DLTSs g_1 and g_2 with $|g_1| + |g_2| \leq k$, as well as the membership constraints, as a SAT instance $SatEnc_k(C)$ (line 3). It then searches for a satisfying assignment and obtains DLTSs g_1 and g_2 based on this assignment (lines 4-8). Since k is increased (line 10) only when the SAT instance is unsatisfiable, minimal DLTSs that satisfy C are obtained.

Algorithm 6.1 Computation of assumptions g_1 and g_2 that satisfy a given set of constraints

```
1:  procedure GENASSMP(C,k)
2:      while 1 do
3:          if SatEnc_k(C) is satisfiable then
4:              Let ψ be a satisfying assignment for SatEnc_k(C)
5:              Let A_1 = A_1(ψ)
6:              Let A_2 = A_2(ψ)
7:              Extend δ_{A_1} and δ_{A_2} to total functions
8:              return (LTS(A_1), LTS(A_2))
9:          end if
10:         Let k = k + 1
11:     end while
12: end procedure
```

6.1 Problem Encoding

We use the following encoding of the problem of finding whether there are DLTSs g_1 and g_2 with k states in total such that $(g_1, g_2) \models C$.

Variables used for encoding the LTSs structure Let k be a number (representing the total number of states in g_1 and g_2) and Let $n = \lceil \log_2(k+2) \rceil$. We use boolean vectors of length n to encode the states of g_1 and g_2, where for each of them we add a special "error" state. Hence, in total we consider $k+2$ states. For each $0 \leq m \leq k+1$ we use $\overline{m}$ to denote the n-bit vector that represents the number m. We fix the vector $\overline{0}$ to represent the error state of g_1, and the vector $\overline{k+1}$ to represent the error state of g_2. We explicitly add the error states in order to distinguish between traces that are rejected by the DLTS and traces for which the behavior is unspecified. For every $i \in \{1, 2\}$:

- Let S_i include the prefixes of all traces over αg_i which are constrained in C with respect to i. That is, $S_i = \bigcup_{c \in C} Strings(c, i)$.
- For every $\sigma \in S_i$, we introduce a set of boolean variables $Var(\sigma, i) = \{v^j_{(\sigma,i)} \mid 0 \leq j \leq n-1\}$. We denote by $\overline{v}_{(\sigma,i)}$ the vector $(v^0_{(\sigma,i)} \cdots v^{n-1}_{(\sigma,i)})$ of boolean variables. $\overline{v}_{(\sigma,i)}$ represents the state of g_i reached when traversing σ.

We define $V_{g_i} = \bigcup_{\sigma \in S_i} Var(\sigma, i)$. In addition to V_{g_1} and V_{g_2}, we introduce a set V_{aux} of boolean variables which consist of the following variables:

- To guarantee that the LTSs we produce are indeed deterministic, we add a set of boolean variables which are used to encode the (non error) transitions in the DLTSs. For this we use $k \times |\alpha g_1 \cup \alpha g_2|$ vectors of boolean variables, each of size n: For every $1 \leq m \leq k$ and $a \in (\alpha g_1 \cup \alpha g_2)$, we introduce a set of boolean variables $Var(m, a) = \{u^j_{(m,a)} \mid 0 \leq j \leq n-1\}$. We denote by $\overline{u}_{(m,a)}$ the vector $(u^0_{(m,a)} \cdots u^{n-1}_{(m,a)})$ of boolean variables. $\overline{u}_{(m,a)}$ represents the state (of either g_1 or g_2) reached from state m after seeing action a.
- To guarantee that the states of the DLTSs are disjoint, we introduce another vector $\overline{u} = (u^0 \cdots u^{n-1})$ of boolean variables, used to represent the number l such that all states of g_1 are smaller than or equal to l and all states of g_2 are larger than l.

Variables used for encoding membership constraints For every disjunctive membership constraint formula $c \in C$ we introduce a boolean "selector" variable en_c that determines which of the disjuncts of c *must* be satisfied (the other disjunct might be satisfied as well). Technically, let En_c be the following set of boolean variables $En_c = \{en_c \mid c \in C\}$, and let $A = En_c \cup \{\neg en_c \mid en_c \in En_c\} \cup \{true\}$.

We define $\theta^{add}_{g_1}, \theta^{rem}_{g_1} : S_1 \to 2^A$ and $\theta^{add}_{g_2}, \theta^{rem}_{g_2} : S_2 \to 2^A$ such that for every $\sigma \in S_i$, $\theta^{add}_{g_i}(\sigma)$ and $\theta^{rem}_{g_i}(\sigma)$ are the smallest sets such that $true \in \theta^{add}_{g_1}(\epsilon)$ and $true \in \theta^{add}_{g_2}(\epsilon)$, and for every $c \in C$:

- if $c = (-(\sigma{\downarrow}_{\alpha g_i}, i) \vee -(\sigma{\downarrow}_{\alpha g_j}, j))$ then $en_c \in \theta^{rem}_{g_i}(\sigma{\downarrow}_{\alpha g_i})$ and $\neg en_c \in \theta^{rem}_{g_j}(\sigma{\downarrow}_{\alpha g_j})$.
- if $c = +(\sigma{\downarrow}_{\alpha g_i}, i)$ then $true \in \theta^{add}_{g_i}(\sigma{\downarrow}_{\alpha g_i})$.
- if $c = -(\sigma{\downarrow}_{\alpha g_i}, i)$ then $true \in \theta^{rem}_{g_i}(\sigma{\downarrow}_{\alpha g_i})$.

- if $c = (-(\sigma{\downarrow}_{\alpha g_j}, j) \vee +(\sigma a{\downarrow}_{\alpha g_i}, i))$ then $en_c \in \theta^{rem}_{g_j}(\sigma{\downarrow}_{\alpha g_j})$ and $\neg en_c \in \theta^{add}_{g_i}(\sigma a{\downarrow}_{\alpha g_i})$.
- if $c = (-(\sigma{\downarrow}_{\alpha g_j}, j) \vee (+(\sigma a{\downarrow}_{\alpha g_i}, i) \wedge -(\sigma a{\downarrow}_{\alpha g_j}, j)))$ then $en_c \in \theta^{rem}_{g_j}(\sigma{\downarrow}_{\alpha g_j})$, $\neg en_c \in \theta^{add}_{g_i}(\sigma a{\downarrow}_{\alpha g_i})$ and $\neg en_c \in \theta^{rem}_{g_j}(\sigma a{\downarrow}_{\alpha g_j})$.

Intuitively, if at least one of the literals in $\theta^{add}_{g_i}(\sigma)$ is satisfied then σ must be added to the language of g_i, and similarly for $\theta^{rem}_{g_i}(\sigma)$ with removal. These sets are therefore interpreted as disjunctions. Formally, let $Bool(A)$ be the set of boolean formulas over A. For $\theta^{ac}_{g_i} : S_i \to 2^A$ (where $ac \in \{rem, add\}$), we define $\tilde{\theta}^{ac}_{g_i} : S_i \to Bool(A)$ as follows:

$$\tilde{\theta}^{ac}_{g_i}(\sigma) = \begin{cases} false & \theta^{ac}_{g_i}(\sigma) = \emptyset \\ \bigvee \theta^{ac}_{g_i}(\sigma) & \text{otherwise} \end{cases}$$

Encoding LTS structure and membership constraints into SAT constraints. $SatEnc_k(C)$ is a set of constraints (with the meaning of conjunction) over the variables $En_c \cup V_{g_1} \cup V_{g_2} \cup V_{aux}$ defined as follows:

- Encoding the LTSs structures into SAT constraints:

  1. For every trace $\sigma_1 \in S_1$ we add the constraint $\overline{v}_{(\sigma_1,1)} \leq \overline{u}$, and for every trace $\sigma_2 \in S_2$ we add the constraint $\overline{u} < \overline{v}_{(\sigma_2,2)}$ (separating states of the DLTSs). We also add a constraint $\overline{1} \leq \overline{u} \leq \overline{k-1}$ to restrict the range of $\overline{u}$.
  2. For every $\sigma \in S_2$ we add the following constraint $\overline{v}_{(\sigma,2)} \leq \overline{k+1}$ (every trace is mapped to a valid state in the DLTSs).
  3. For every $i \in \{1, 2\}$, every trace $\sigma \in S_i$, every action $a \in \alpha g_i$ such that $\sigma a \in S_i$, and for every $1 \leq m \leq k$, we add the following constraint: $\overline{v}_{(\sigma,i)} = \overline{m} \Rightarrow \overline{v}_{(\sigma a,i)} = \overline{u}_{(m,a)}$ (the DLTSs are deterministic).
  4. For every trace $\sigma \in S_1$ and action $a \in \alpha g_1$, if $\sigma a \in S_1$ then we add the following constraint: $\overline{v}_{(\sigma,1)} = \overline{0} \Rightarrow \overline{v}_{(\sigma a,1)} = \overline{0}$ (the error state of g_1 is a sink state; DLTSs are prefix closed).
  5. For every string $\sigma \in S_2$ and action $a \in \alpha g_2$, if $\sigma a \in S_2$ then we add the following constraint: $\overline{v}_{(\sigma,2)} = \overline{k+1} \Rightarrow \overline{v}_{(\sigma a,2)} = \overline{k+1}$ (the error state of g_2 is a sink state; DLTSs are prefix closed).

  Remark 1. Item 1 ensures that for every trace $\sigma_1 \in S_1$ and for every trace $\sigma_2 \in S_2$, $\overline{v}_{(\sigma_1,1)} \neq \overline{v}_{(\sigma_2,2)}$. Encoding the latter requires $O(|S_1| \times |S_2|)$ constraints, whereas item 1 defines only $O(|S_1|+|S_2|)$ constraints, with the use of V_{aux}. Similarly, item 3 ensures that $(\overline{v}_{(\sigma,i)} = \overline{v}_{(\sigma',i)} \Rightarrow \overline{v}_{(\sigma a,i)} = \overline{v}_{(\sigma' a,i)})$. Again, a direct encoding requires $O(|\alpha g_i| \times |S_i|^2)$ constraints, whereas the indirect encoding used in item 3 defines only $O(k \times |\alpha g_i| \times |S_i|)$ constraints (where typically $k << |S_i|$).

- Encoding the membership constraints formulas into SAT constraints:

6. For every trace $\sigma \in S_1$ we add the constraint: $\tilde{\theta}^{rem}_{g_1}(\sigma) \Rightarrow \overline{v}_{(\sigma,1)} = \overline{0}$.
7. For every trace $\sigma \in S_2$ we add the constraint: $\tilde{\theta}^{rem}_{g_2}(\sigma) \Rightarrow \overline{v}_{(\sigma,2)} = \overline{k+1}$.
8. For every trace $\sigma \in S_1$ we add the constraint: $\tilde{\theta}^{add}_{g_1}(\sigma) \Rightarrow \overline{v}_{(\sigma,1)} \neq \overline{0}$.
9. For every trace $\sigma \in S_2$ we add the constraint: $\tilde{\theta}^{add}_{g_2}(\sigma) \Rightarrow \overline{v}_{(\sigma,2)} \neq \overline{k+1}$.

Note that the implications in constraints 6-9 guarantee that a trace is accepted by g_i (leads to a non-error state) whenever it is required to be added to g_i (as encoded by $\theta^{add}_{g_i}(\sigma{\downarrow_{\alpha g_i}})$). However, it may be accepted also in other cases, provided it is not required to be removed by other constraints. The same holds for removal of traces from g_i.

Optimized implementation. When k does not change, and only C increases, the SAT encoding is incremental, as we only add constraints (in particular, the change in constraints 6-9 is encoded using additional clauses). In order to support incremental SAT calls also when k changes into k' such that $\lceil \log_2(k+2) \rceil = \lceil \log_2(k'+2) \rceil$, we turn all constraints that refer to $\overline{k+1}$ or $\overline{k-1}$ to conditional, guarded by some Boolean variable. When k increases into k' we then "cancel" the clauses that refer to $\overline{k+1}$ and $\overline{k-1}$ and replace them by clauses that refer to $\overline{k'+1}$ or $\overline{k'-1}$, respectively. $\overline{k+1}$ is used in $O(|S_2|)$ constraints. In our implementation, in order to minimize the number of conditional constraints, we therefore use another vector $\overline{u}_{max} = (u^0_{max} \cdots u^{n-1}_{max})$ of Boolean variables instead of $\overline{k+1}$ in all constraints, and add a (single) conditional constraint $\overline{u}_{max} = \overline{k+1}$, guarded by another Boolean variable. The desired values of the Boolean variables in the guards of the conditional constraints are sent as assumptions to the SAT solver.

Lemma 6.1. *$SatEnc_k(C)$ is satisfiable if and only if there exist DLTSs g_1 and g_2 that satisfy C such that $|g_1| + |g_2| = k$.*

Proof. ($\Longrightarrow$:) We defer the proof of the implication from the left to the right to section 6.2, where we show how to construct LTSs from a satisfying assignment of $SatEnc_k(C)$.

($\Longleftarrow$:) Let g_1 and g_2 be two DLTSs that satisfy C such that $|g_1| + |g_2| = k$. We show that $SatEnc_k(C)$ is satisfiable. We extend g_1 by a sink error state denoted π_1, and similarly extend g_2 by a sink error state denoted π_2.

Let $\phi_{g_1} : S_{g_1} \to \{0, \ldots, (|g_1| - 1)\}$ and $\phi_{g_2} : S_{g_2} \to \{|g_1|, \ldots, (|g_1| + |g_2| - 1)\}$ be two bijective functions over the (extended) states of g_1 and g_2, respectively. We assume also that ϕ_{g_1} maps π_1 to 0 and ϕ_{g_2} maps π_2 to $k+1$ (such functions can be easily constructed). Let ψ be the following assignment:

- $\forall en_c \in En_c$:

$$\psi(en_c) = \begin{cases} \sigma{\downarrow} \in L(g_1)\ ?\ 0 : 1 & \text{if } c = -(\sigma{\downarrow_{\alpha g_1}}, 1) \vee -(\sigma{\downarrow_{\alpha g_2}}, 2) \\ \sigma{\downarrow} \in L(g_1)\ ?\ 0 : 1 & \text{if } c = -(\sigma{\downarrow_{\alpha g_1}}, 1) \vee +(\sigma a{\downarrow_{\alpha g_2}}, 2) \\ \sigma{\downarrow} \in L(g_2)\ ?\ 0 : 1 & \text{if } c = -(\sigma{\downarrow_{\alpha g_2}}, 2) \vee +(\sigma a{\downarrow_{\alpha g_1}}, 1) \\ \sigma{\downarrow} \in L(g_1)\ ?\ 0 : 1 & \text{if } c = -(\sigma{\downarrow_{\alpha g_1}}, 1) \vee (+(\sigma a{\downarrow_{\alpha g_2}}, 2) \wedge -(\sigma a{\downarrow_{\alpha g_1}}, 1)) \\ \sigma{\downarrow} \in L(g_2)\ ?\ 0 : 1 & \text{if } c = -(\sigma{\downarrow_{\alpha g_2}}, 2) \vee (+(\sigma a{\downarrow_{\alpha g_1}}, 1) \wedge -(\sigma a{\downarrow_{\alpha g_2}}, 2)) \end{cases}$$

- $\forall s \in S_1 : \psi(\overline{v}_{(s,1)}) = \phi_{g_1}(\delta_{g_1}(s_0^{g_1}, s))$
- $\forall s \in S_2 : \psi(\overline{v}_{(s,2)}) = \phi_{g_2}(\delta_{g_2}(s_0^{g_2}, s))$
- $\psi(\overline{u}) = |g_1| - 1$
- $\forall 1 \leq m \leq k : \psi(\overline{u}_{(m,a)}) = \begin{cases} \phi_{g_1}(\delta_{g_1}(\phi_{g_1}^{-1}(m), a)) & \text{if } m \leq |g_1| - 1 \\ \phi_{g_2}(\delta_{g_2}(\phi_{g_2}^{-1}(m), a)), & \text{otherwise} \end{cases}$

Note that by the definition of ψ, $\psi(\overline{v}_{(\sigma,1)}) = \overline{0}$ implies that σ is not in $L(g_1)$ and similarly $\psi(\overline{v}_{(\sigma,2)}) = \overline{k+1}$ implies that σ is not in $L(g_2)$. We now show that ψ is satisfying assignment for $SatEnc_k(C)$ by showing that ψ satisfies all constraints that are in $SatEnc_k(C)$:

- Constraints numbers 1- 2 in $SatEnc_k(C)$ are trivially satisfied by the definitions of ψ, ϕ_{g_1} and ϕ_{g_2}.
- Constraint number 3: For every i in $\{1, 2\}$, every $1 \leq m \leq k$, every σ in S_i, and for every a in αg_i such that $\sigma a \in S_i$, if $\psi(\overline{v}_{(\sigma,i)})$ is equal to m it implies by the definition of ψ that $\delta_{g_i}(s_0^{g_i}, \sigma) = \phi_{g_i}^{-1}(m)$. Since δ_{g_i} is deterministic it follows that (1) $\delta_{g_i}(s_0^{g_i}, \sigma a)$ is equal to $\delta_{g_i}(\phi_{g_i}^{-1}(m), a)$. By (1) and by the definition of ψ we get that $\psi(\overline{v}_{(\sigma a,i)})$ is equal to $\phi_{g_i}(\delta_{g_i}(\phi_{g_i}^{-1}(m), a))$. On the other hand, by the definition of ϕ_{g_i} and ψ we know that $\psi(\overline{u}_{(m,a)})$ is equal to $\phi_{g_i}(\delta_{g_i}(\phi_i^{-1}(m), a))$. Hence, we get that $\psi(\overline{v}_{(\sigma a,i)})$ is equal to $\psi(\overline{u}_{(m,a)})$.
- Constraint number 4: For every σ in S_1, $\psi(\overline{v}_{(\sigma,1)}) = \overline{0}$ implies that $\phi_{g_1}(\delta_{g_1}(s_0^{g_1}, \sigma)) = 0$ therefore, $\delta_{g_1}(s_0^{g_1}, \sigma) = \pi_1$, which means that (1) σ is not in $L(g_1)$. For every a in αg_1, σa is is not in $L(g_1)$ (By (1) and by the fact that π_1 is an error sink state), which implies that $\delta_{g_1}(s_0^{g_1}, \sigma a) = \pi_1$. By the definition of the assignment ψ we get that $\psi(\overline{v}_{(\sigma a,1)}) = \phi_{g_1}(\delta_{g_1}(s_0^{g_1}, \sigma a)) = \phi_{g_1}(\pi_1)$. Hence, we get that $\psi(\overline{v}_{(\sigma a,1)})$ is equal to $\overline{0}$.
- Constraint number 5: The proof is similar to the proof of constraint number 4.
- Constraint number 6: For every σ in S_1, $\tilde{\theta}_{g_1}^{rem}(\sigma)$ equals 1 implies that one of following three conditions hold:
  - $\tilde{\theta}_{g_1}^{rem}(\sigma) \equiv true$, which implies that there exists c in C such that $c = -(\sigma{\downarrow}_{\alpha g_1}, 1)$. Since (g_1, g_2) satisfies C we get that (g_1, g_2) satisfies c as well. It implies that $\sigma{\downarrow}_{\alpha g_1}$ is not in $L(g_1)$. Therefore, by the definitions of ψ and ϕ_{g_1} we get that $\psi(\overline{v}_{(\sigma,1)})$ is equal to $\overline{0}$.
  - $\tilde{\theta}_{g_1}^{rem}(\sigma) = en_c \vee \theta_{g_1}(\sigma)$ and $\psi(en_c) = 1$ for some constraint c and some boolean function $\theta_{g_1}(\sigma)$. Then, c is one of the following forms:
    - $c = -(\sigma{\downarrow}_{\alpha g_1}, 1) \vee -(\sigma{\downarrow}_{\alpha g_2}, 2)$.
    - $c = -(\sigma{\downarrow}_{\alpha g_1}, 1) \vee +(\sigma a{\downarrow}_{\alpha g_2}, 2)$.
    - $c = -(\sigma{\downarrow}_{\alpha g_1}, 1) \vee (+(\sigma a{\downarrow}_{\alpha g_2}, 2) \wedge -(\sigma a{\downarrow}_{\alpha g_1}, 1))$.

For all these forms of constraints, $\psi(en_c)$ equals 1 implies by the definition of ψ that σ is not in $L(g_1)$. Thus, by (1) we get that $\psi(\overline{v}_{(\sigma,i)})$ is equal to $\overline{0}$.

– $\tilde{\theta}^{rem}_{g_1}(\sigma) = \neg en_c \vee \theta_{g_1}(\sigma)$ and $\psi(en_c) = 0$ for some constraint c and some boolean function $\theta_{g_1}(\sigma)$ where $\sigma = \sigma' a$. Then, c must be of the following form:

  * $c = -(\sigma'\downarrow_{\alpha g_1}, 1) \vee (+(\sigma' a\downarrow_{\alpha g_2}, 2) \wedge -(\sigma' a\downarrow_{\alpha g_1}, 1))$

  For such a constraint c, by the definition of the assignment ψ, it follows that $\psi(en_c)$ equals 0 implies that $\sigma'\downarrow_{\alpha g_1}$ is in $L(g_1)$. Therefore, since $(g1, g2)$ satisfies c, it follows that $\sigma' a\downarrow_{\alpha g_2}$ is in $L(g_2)$ and $\sigma' a\downarrow_{\alpha g_1}$ is not in $L(g_1)$, which implies that $\psi(\overline{v}_{(\sigma,1)}) = \phi_{g_1}(\delta_{g_1}(s^{g_1}_0, \sigma)) = \phi_{g_1}(\pi_1) = 0$.

To summarize, we have shown that for every σ in S_1, $\tilde{\theta}^{rem}_{g_1}(\sigma)$ equals 1 implies that $\psi(\overline{v}_{(\sigma,1)})$ equals $\overline{0}$. Hence, the assignment ψ satisfies constraint number 6.

Constraint number 7: The proof is similar to the proof of constraint number 6.

Constraint number 8: For every σ in S_1, $\tilde{\theta}^{add}_{g_1}(\sigma)$ equals 1 implies that one of following two conditions hold:

- $\tilde{\theta}^{add}_{g_1}(\sigma) \equiv true$. Therefore, there exists c in C such that c equals $+(\sigma\downarrow_{\alpha g_1}, 1)$. Since (g_1, g_2) satisfies C, it follows that (g_1, g_2) satisfies c. Thus, $\sigma\downarrow_{\alpha g_1}$ is in $L(g_1)$. Hence, by the definition of ψ and ϕ_{g_1} we get that $\psi(\overline{v}_{(\sigma,1)})$ is not equal to $\phi_{g_1}(\pi_1)$, which equals 0.
- $\tilde{\theta}^{add}_{g_1}(\sigma) = \neg en_c \vee \theta_{g_i}(\sigma)$ and $\psi(en_c) = 0$ for some constraint c and some boolean function $\theta_{g_1}(\sigma)$ where $\sigma = \sigma' a$. Then, c is one of the following forms:

  – $c = -(\sigma'\downarrow_{\alpha g_2}, 2) \vee +(\sigma' a\downarrow_{\alpha g_1}, 1)$.

  – $c = -(\sigma'\downarrow_{\alpha g_2}, 2) \vee (+(\sigma' a\downarrow_{\alpha g_1}, 1) \wedge -(\sigma' a\downarrow_{\alpha g_2}, 2))$.

  For all these forms of constraints, by the definition of the assignment ψ, it follows that $\psi(en_c)$ equals 0 implies that $\sigma\downarrow_{\alpha g_2}$ is in $L(g_2)$. Therefore, since (g_1, g_2) satisfies C we get that $\sigma a\downarrow_{\alpha g_1}$ is in $L(g_1)$. Hence, we get that $\psi(\overline{v}_{(\sigma,1)})$ is not equal to $\overline{0}$.

  To summarize, we have shown that for every σ in S_1, $\tilde{\theta}^{add}_{g_1}(\sigma)$ equals 1 implies that σ is in $L(g_1)$, which implies that $\psi(v_{(\sigma,1)})$ is not equal to $\overline{0}$.

- Constraint number 9: The proof is similar to the proof of constraint number 8 □

Due to Lemma 5.5 which ensures that (the nondeterministic) LTSs $M_1\downarrow_{\alpha g_1}$ and $M_2\downarrow_{\alpha g_2}$ satisfy C, we get the following corollary, which ensures termination of GENASSMP:

Corollary 6.2. *At every iteration of* ACR*, there exists* $k \leq O(2^{|M_1\downarrow_{\alpha g_1}|} + 2^{|M_2\downarrow_{\alpha g_2}|})$ *where* $SatEnc_k(C)$ *is satisfiable.*

Proof. By Lemma 5.5 we know that $(M_1\downarrow_{\alpha g_1}, M_2\downarrow_{\alpha g_2})$ satisfy any set of constraints C. By determinizing $M_1\downarrow_{\alpha g_1}$ and $M_2\downarrow_{\alpha g_2}$ we get DLTSs whose number of states are $O(2^{|M_1\downarrow_{\alpha g_1}|})$ and $O(2^{|M_2\downarrow_{\alpha g_2}|})$ respectively. Then by Lemma 6.1 we get that at any iteration of Algorithm 4.2 there exists $k = O(2^{|M_1\downarrow_{\alpha g_1}|} + 2^{|M_2\downarrow_{\alpha g_2}|})$ for which $SatEnc_k(C)$ is SAT. □

In fact, since satisfiability of $SatEnc_k(C)$ is checked for increasing values of k, it is ensured that the minimal k for which it is satisfiable is found. Therefore, minimal assumptions that satisfy C are obtained. In particular, together with Lemma 5.4, this ensures that when $M_1||M_2 \models P$, then minimal assumptions for which CIRC-AG is applicable are eventually obtained.

6.2 From SAT Assignment to LTS Assumptions

Given a satisfying assignment ψ to $SatEnc_k(C)$, lines 4-8 of Algorithm 6.1 uses the assignment ψ to generate assumptions g_1 and g_2 that satisfy C, as described below.

First, in lines 5-6 we extract DLTSs $A_1(\psi)$ and $A_2(\psi)$ extended with error states. $A_1(\psi)$ and $A_2(\psi)$ can be thought of as error LTSs, except that they might be incomplete. As in an error LTS, traces leading to an error state in $A_i(\psi)$ are rejected. Traces that have no corresponding path are unspecified (recall that such traces do not exist in an error LTS, which is complete, and in a DLTS, in contrast, such traces are rejected). The latter represent traces that do not affect the satisfaction of C, and can therefore either be accepted or rejected.

Definition 6.3. Let ψ be a satisfying assignment for $SatEnc_k(C)$. We define $A_1(\psi)$ and $A_2(\psi)$ derived from ψ in the following way: $A_i(\psi) = (Q_i, \alpha g_i, \delta_i, q_0^i, \pi_i)$ where:

- $Q_i = \{\overline{m} \in \{0,1\}^n \mid \exists \sigma \in S_i \text{ such that } \psi(\overline{v}_{(\sigma,i)}) = \overline{m}\}$
- $\delta_i(\overline{m}, a) = \begin{cases} \overline{m}' & \text{if } \exists \sigma \in S_i \text{ such that } \psi(\overline{v}_{(\sigma,i)}) = \overline{m} \wedge \sigma a \in S_i \wedge \psi(\overline{v}_{(\sigma a,i)}) = \overline{m}' \\ \perp & \text{otherwise} \end{cases}$
- $q_0^i = \psi(\overline{v}_{(\epsilon,i)})$
- $\pi_1 = \overline{0}$
- $\pi_2 = \overline{k+1}$

Note that δ_i is deterministic and it is well defined, since constraint 3 of $SatEnc_k(C)$ ensures that if there exist $\sigma, \sigma' \in S_i$ such that $\psi(\overline{v}_{(\sigma,i)}) = \psi(\overline{v}_{(\sigma',i)})$ and both σa and $\sigma' a$ are in S_i, then also $\psi(\overline{v}_{(\sigma a,i)}) = \psi(\overline{v}_{(\sigma' a,i)})$. Note further, that due to constraint 1, $Q_1 \cap Q_2 = \emptyset$.

δ_1 and δ_2 are partial functions. In line 7 of Algorithm 6.1 we extend δ_1 and δ_2 to be total functions, transforming $A_1(\psi)$ and $A_2(\psi)$ into (complete) error LTSs. As explained above, the cases in which δ_1 and δ_2 are undefined are cases that do not affect satisfaction of C. Therefore, any completion will result in DLTSs that satisfy C. In practice, we extend δ_1 and δ_2 to be total functions in the following way: If the transition from a given state $q \in Q_i$ on action a is undefined in the transition relation, we add a self loop for the state q labeled by a, i.e., we define $\delta_i(q, a) = q$.

In order to obtain DLTSs, it remains to remove the error states. In line 8 of Algorithm 6.1, for every i in $\{1, 2\}$ we compute $LTS(A_i)$ defined as follows:

Definition 6.4. Let $A_i = (Q_i, \alpha g_i, \delta_i, q_0^i, \pi_i)$ be defined as in Definition 6.3, with δ_i extended as described above. Then $LTS(A_i)$ is the DLTS $(Q_i \setminus \{\pi_i\}, \alpha g_i, \delta_i', q_0^i)$ where

$$\delta_i'(q,a) = \begin{cases} \delta_i(q,a) & \text{if } \delta_i(q,a) \neq \pi_i \\ \bot & \text{otherwise} \end{cases}$$

The following technical lemma states that the DLTSs obtained from definition 6.4 agree with ψ on traces in S_i that lead to a non-error state, and reject traces of S_i that lead to an error state.

Lemma 6.5. *For every $i \in \{1,2\}$ and for every trace $\sigma \in S_i$, we have that:*

$$\delta_i'(q_0^i, \sigma) = \begin{cases} \psi(\overline{v}_{(\sigma,i)}) & \textit{if } \psi(\overline{v}_{(\sigma,i)}) \neq \pi_i \\ \bot & \textit{otherwise} \end{cases}$$

Proof. We prove the lemma by induction on the length of σ.
Base case: $|\sigma| = 0$ i.e. $\sigma = \epsilon$. Since $true \in \theta_{g_1}^{add}(\epsilon)$, based on constraint 8 and 9 of $SatEnc_k(C)$ we have that $\psi(\overline{v}_{(\epsilon,i)}) \neq \pi_i$. We therefore need to show that $\delta_i'(q_0^i, \epsilon) = \psi(\overline{v}_{(\epsilon,i)})$. This holds since $\delta_i'(q_0^i, \epsilon) = q_0^i$ and since $q_0^i = \psi(\overline{v}_{(\epsilon,i)})$ (by the definition of q_0^i in $A_i(\psi)$).
Induction step: $|\sigma| \geq 1$. Let σ be $\sigma' a$ where $a \in \alpha g_i$. Note that since S_i contains all the prefixes of σ, we get that $\sigma' \in S_i$ as well. Since both $\sigma' \in S_i$ and $\sigma = \sigma' a \in S_i$, the definition of δ_i in $A_i(\psi)$ ensures that $\delta_i(\psi(\overline{v}_{(\sigma',i)}), a) = \psi(\overline{v}_{(\sigma' a,i)}) = \psi(\overline{v}_{(\sigma,i)})$. We now distinguish between the two possibilities:

- $\psi(\overline{v}_{(\sigma,i)}) \neq \pi_i$: Recall that $\delta_i(\psi(\overline{v}_{(\sigma',i)}), a) = \psi(\overline{v}_{(\sigma,i)})$. Since $\psi(\overline{v}_{(\sigma,i)}) \neq \pi_i$, we have that $\delta_i'(\psi(\overline{v}_{(\sigma',i)}), a) = \psi(\overline{v}_{(\sigma,i)})$ as well (see Definition 6.4). Based on constraints 4 and 5 of $SatEnc_k(C)$ we have that $\psi(\overline{v}_{(\sigma',i)}) \neq \pi_i$ (since $\psi(\overline{v}_{(\sigma,i)}) \neq \pi_i$). Therefore, by the induction hypothesis on $\sigma' \in S_i$, we have that $\delta_i'(q_0^i, \sigma') = \psi(\overline{v}_{(\sigma',i)})$. Finally, since $LTS(A_i)$ is deterministic and since $\delta_i'(q_0^i, \sigma') \neq \bot$, we have that $\delta_i'(q_0^i, \sigma) = \delta_i'(\delta_i'(q_0^i, \sigma'), a) = \delta_i'(\psi(\overline{v}_{(\sigma',i)}), a) = \psi(\overline{v}_{(\sigma,i)})$, as required.

- $\psi(\overline{v}_{(\sigma,i)}) = \pi_i$: Again, recall that $\delta_i(\psi(\overline{v}_{(\sigma',i)}), a) = \psi(\overline{v}_{(\sigma,i)})$, i.e., $\delta_i(\psi(\overline{v}_{(\sigma',i)}), a) = \pi_i$. Therefore, by Definition 6.4, $\delta_i'(\psi(\overline{v}_{(\sigma',i)}), a) = \bot$. Now, since $LTS(A_i)$ is deterministic, we get that if $\delta_i'(q_0^i, \sigma') = \bot$, then $\delta_i'(q_0^i, \sigma) = \delta_i'(q_0^i, \sigma') = \bot$, and if $\delta_i'(q_0^i, \sigma') \neq \bot$ then $\delta_i'(q_0^i, \sigma) = \delta_i'(\delta_i'(q_0^i, \sigma'), a) = \bot$. Either way, $\delta_i'(q_0^i, \sigma) = \bot$, as required. □

Lemma 6.6. *Let ψ be a satisfying assignment for $SatEnc_k(C)$, let $g_1 = LTS(A_1(\psi))$ and $g_2 = LTS(A_2(\psi))$. Then g_1 and g_2 are DLTSs such that (1) $(g_1, g_2) \models C$ and (2) $|g_1| + |g_2| \leq k$.*

Proof. In order to prove that $(LTS(A_1(\psi)), LTS(A_2(\psi))$ satisfies C we need to show that $(LTS((A_1(\psi)), LTS(A_2(\psi)))$ satisfies every constraint c in C. For every (i,j) in $\{(1,2),(2,1)\}$, a constraint c in C can have one of the following forms:

- $c = -(\sigma{\downarrow}_{\alpha g_i}, 1)) \vee -(\sigma{\downarrow}_{\alpha g_j}, 2)$ implies by the definitions of $\theta^{rem}_{g_1}(\sigma{\downarrow}_{\alpha g_1})$ and $\theta^{rem}_{g_2}(\sigma{\downarrow}_{\alpha g_2})$, that $\psi(\theta^{rem}_{g_1}(\sigma{\downarrow}_{\alpha g_1}))$ is true or $\psi(\theta^{rem}_{g_2}(\sigma{\downarrow}_{\alpha g_2}))$ is true. Therefore by constraints number 6 and 7 we get that $\psi(\overline{v}_{(\sigma{\downarrow}_{\alpha g_1},1)})$ is equal to π_1 or $\psi(\overline{v}_{(\sigma{\downarrow}_{\alpha g_2},2)})$ is equal to π_2. Hence, by Lemma 6.5 we get that $\sigma{\downarrow}_{\alpha g_1}$ is not in $L(LTS(A_1(\psi)))$ or $\sigma{\downarrow}_{\alpha g_2}$ is not in $L(LTS(A_2(\psi)))$, which means that $(LTS(A_1(\psi)), LTS(A_2(\psi)))$ satisfies c.

- $c = +(\sigma{\downarrow}_{\alpha g_i}, i)$ implies by the definition of $\theta^{add}_{g_i}(\sigma{\downarrow}_{\alpha g_i})$, that $\psi(\theta^{add}_{g_i}(\sigma{\downarrow}_{\alpha g_i}))$ is true. Therefore by constraints number 8 and 9, we get that $\psi(\overline{v}_{(\sigma{\downarrow}_{\alpha g_i},i)})$ is not equal to π_i. Thus, by Lemma 6.5 it follows that $\sigma{\downarrow}_{\alpha g_i}$ is in $L(LTS(A_i(\psi)))$. Hence, $(LTS(A_1(\psi)), LTS(A_2(\psi)))$ satisfies c.

- $c = -(\sigma{\downarrow}_{\alpha g_i}, i)$ implies by definition of $\theta^{rem}_{g_i}(\sigma{\downarrow}_{\alpha g_i})$ that $\psi(\theta^{rem}_{g_i}(\sigma{\downarrow}_{\alpha g_i}))$ is true. Therefore, by constraints number 6 and 7 we get that $\psi(\overline{v}_{(\sigma{\downarrow}_{\alpha g_i},i)})$ is equal to π_i. Hence, by Lemma 6.5 we get that $\sigma{\downarrow}_{\alpha g_i}$ is not in $L(LTS(A_i(\psi)))$, which implies that $(LTS(A_1(\psi)), LTS(A_2(\psi)))$ satisfies c.

- $c = -(\sigma{\downarrow}_{\alpha g_j}, j) \vee +(\sigma a{\downarrow}_{\alpha g_i}, i)$ implies by the definitions of $\theta^{rem}_{g_j}(\sigma{\downarrow}_{\alpha g_j})$ and $\theta^{add}_{g_i}(\sigma{\downarrow}_{\alpha g_i})$, that $\psi(\theta^{rem}_{g_j}(\sigma{\downarrow}_{\alpha g_j}))$ is true or $\psi(\theta^{add}_{g_i}(\sigma a{\downarrow}_{\alpha g_i}))$ is true. As a result, constraints number 6 to 9 imply that $\psi(\overline{v}_{(\sigma{\downarrow}_{\alpha g_j},j)})$ is not equal to π_j or $\psi(\overline{v}_{(\sigma a{\downarrow}_{\alpha g_i},i)})$ is not equal to π_i. Hence, by Lemma 6.5 we get that $\sigma{\downarrow}_{\alpha g_j}$ is not in $L(LTS(A_j(\psi)))$ or that $\sigma a{\downarrow}_{\alpha g_i}$ is in $L(LTS(A_i(\psi)))$, which means that $(LTS(A_1(\psi)), LTS(A_2(\psi)))$ satisfies c.

- $c = -(\sigma{\downarrow}_{\alpha g_j}, j) \vee (+(\sigma a{\downarrow}_{\alpha g_i}, i) \wedge (\sigma a{\downarrow}_{\alpha g_j}, j))$ implies by the definitions of $\theta^{rem}_{g_j}(\sigma{\downarrow}_{\alpha g_j}))$ and $\theta^{add}_{g_i}(\sigma{\downarrow}_{\alpha g_i}))$ that $\psi(\theta^{rem}_{g_j}(\sigma{\downarrow}_{\alpha g_j}))$ is true or both $\psi(\theta^{add}_{g_i}(\sigma a{\downarrow}_{\alpha g_i})$ and $\psi(\theta^{rem}_{g_j}(\sigma a{\downarrow}_{\alpha g_j}))$ are true. Therefore, by constraints number 6 to 9 it follows that $\psi(\overline{v}_{(\sigma{\downarrow}_{\alpha g_j},j)})$ is equal to π_j or both $\psi(\overline{v}_{(\sigma a{\downarrow}_{\alpha g_i},i)})$ is not equal to π_i and $\psi(\overline{v}_{(\sigma a{\downarrow}_{\alpha g_j},j)})$ is equal to π_j. Hence, by Lemma 6.5 we get that $\sigma{\downarrow}_{\alpha g_j}$ is not in $L(LTS(A_j(\psi)))$ or $\sigma a{\downarrow}_{\alpha g_i}$ is in $L(LTS(A_i(\psi)))$ and $\sigma a{\downarrow}_{\alpha g_j}$ is not in $L(LTS(A_j(\psi)))$, which means that $(LTS(A_1(\psi)), LTS(A_2(\psi)))$ satisfies c. □

Now we show that $|LTS(A_1(\psi))| + |LTS(A_2(\psi)) \leq k$: (i) We know that ψ satisfies $SatEnc_k(C)$, therefore by constraints number 1 and 2 we get that for every v in V_{g_1}, V_{g_2}, $\psi(v)$ is less than or equals to $k-1$. This implies that Q_1 and Q_2 are subsets of $\{0 \cdots k-1\}$. (ii) By constraint number 1 we get that for every v_1 in V_{g_1} and every v_2 in V_{g_2}, $\psi(v_1)$ is not equal to $\psi(v_2)$. It follows that Q_1 and Q_2 do not intersect. From (i) and (ii) we get that $|Q_1| + |Q_2| \leq k$. Hence, $|A_1(\psi)| + |A_2(\psi)| \leq k$ □

Combined with Corollary 5.3, we also conclude that:

Corollary 6.7. *The DLTSs $g_1 = LTS(A_1(\psi))$ and $g_2 = LTS(A_2(\psi))$ generated by Algorithm 6.1 are different from all the pairs of DLTSs considered in previous iterations.*

Example 5. Consider the 7th (and final) iteration of ACR on the example from section 5.2. Since the assumptions from the 6th iteration (Figure 5.1) have a total of 3 states, the search performed

by GENASSMP at the 7th iteration starts with $k = 3$, and since no satisfying assignment is found for $SatEnc_3(C)$, k is increased to 4, yielding g_1 and g_2 with a total of 4 states (Figure 3.1). Note that the (final) assumptions g_1 and g_2 generated by GENASSMP in the 7th iteration indeed satisfy the membership constraint $-(\langle send \rangle, 1) \vee (+(\langle send, send \rangle, 2) \wedge -(\langle send, send \rangle, 1)) \in C$ from the previous iteration (due to the right disjunct). In particular, they do not exhibit the counterexample from Example 4.

To complete this chapter, we complete the proof of lemma 6.1:

Proof of lemma 6.1 (cont.). ($\Longrightarrow$:) By Lemma 6.6, there exist g_1 and g_2 that satisfy C such that $|g_1| + |g_2| \leq k$. It remains to show that there exist g_1 and g_2 such that $k = |g_1| + |g_2|$ as needed in lemma 6.1. In the case where $|g_1| + |g_2| < k$, to get g_1 and g_2 such that $k = |g_1| + |g_2|$ we add some unreachable states to g_1 or to g_2. □

Chapter 7

Correctness, Termination and Minimality

In this chapter we argue that our main algorithm ACR is correct, it terminates and produces minimal assumptions.

Theorem 7.1 (Correctness and Termination)**.** *Given components M_1 and M_2, and property P,* ACR *terminates and returns "$M_1||M_2 \models P$" if P holds on $M_1||M_2$ and "$M_1||M_2 \not\models P$", otherwise.*

Proof. Partial Correctness: Algorithm 4.2 returns $M_1||M_2 \models P$ if and only if Algorithm 5.1 returns $M_1||M_2 \models P$. The latter returns $M_1||M_2 \models P$ if and only if g_1 and g_2 satisfy all the three premises of CIRC-AG. Thus, by the soundness of CIRC-AG from Theorem 3.6, we get that Algorithm 4.2 returns $M_1||M_2 \models P$ only if $M_1||M_2$ satisfies P. On the other hand Algorithm 4.2 returns that $M_1||M_2 \not\models P$ if and only if Algorithm 5.1 returns so. The latter may return $M_1||M_2 \not\models P$ in line 5 of Algorithm 5.2, in line 11 of Algorithm 5.1 and in line 5 of Algorithm 5.1. All these cases are conditioned by the existence of σ in $(\alpha g_1 \cup \alpha g_2)^*$ such that σ is in $L(M_1{\downarrow_{\alpha g_1}}||M_2{\downarrow_{\alpha g_2}})$ but ${\sigma\downarrow_{\alpha P}}$ is not in P, which implies that $M_1{\downarrow_{\alpha g_1}}||M_2{\downarrow_{\alpha g_2}}$ does not satisfy P. Therefore, by Lemma 3.5 we get that $M_1||M_2$ does not satisfy P.

Termination: (1) By Corollary 6.2 we get that, at any iteration of Algorithm 4.2, there exists $k = O(2^{|M_1{\downarrow_{\alpha g_1}}|} + 2^{|M_2{\downarrow_{\alpha g_2}}|})$ where $SatEnc_k(C)$ is satisfiable. Since in GENASSMP, k is increased when $SatEnc_k(C)$ is unsatisfiable, it is guaranteed that a satisfying assignment ψ will be found. (2) By Lemma 6.6 and by Corollary 6.7 we get that at each iteration of Algorithm 4.2 we get a different pair of DLTSs $A_1(\psi)$ and $A_2(\psi)$ that satisfy C and their total size is less than k. Since there are only finitely many pairs of DTLS for every k, in the worst case, we eventually get to $k = |det(M_1{\downarrow_{\alpha g_1}})| + |det(M_2{\downarrow_{\alpha g_2}})| = O(2^{|M_1{\downarrow_{\alpha g_1}}|} + 2^{|M_2{\downarrow_{\alpha g_2}}|})$ and get the pair of DLTSs $det(M_1{\downarrow_{\alpha g_1}})$ and $det(M_2{\downarrow_{\alpha g_2}})$ that by Lemma 5.5 satisfy C and have total number of states that is less than k. For this pair, Algorithm 5.1 is guaranteed to return either "$M_1||M_2 \models P$" or "$M_1||M_2 \not\models P$". Hence, Algorithm 4.2 is guaranteed to terminate. □

Theorem 7.2 (Minimality)**.** *If* $M_1||M_2 \models P$ *then* ACR *terminates with DLTSs* g_1 *and* g_2 *whose total number of states is minimal among all pairs of DLTSs that satisfy the CIRC-AG rule.*

Proof. Theorem 7.1 already ensures that Algorithm 4.2 will terminate with the result "$M_1||M_2 \models P$". It remains to prove the minimality of the obtained assumptions. By Lemma 5.4, we get that the DLTSs with minimum total number of states that satisfy rule CIRC-AG satisfy any set of constraints C that is being produced by Algorithm 5.1. We denote by n the total number of states in the DLTSs with minimum total number of states that satisfy rule CIRC-AG. By applying Lemma 6.1 we get that $SatEnc_n(C)$ is satisfiable. Since $SatEnc_n(C)$ is satisfiable, k never gets to be greater than n. By Lemma 6.6 we get that the DLTSs pair $(A_1(\psi), A_2(\psi))$ of any satisfying assignment ψ of $SatEnc_n(C)$ has a total number of states which is less than or equal to n. In particular this holds for the final assumptions for which Algorithm 5.1 returns "$M_1||M_2 \models P$", and the claim follows. □

Chapter 8

Evaluation

We implemented ACR in the LTSA (Labelled Transition System Analyser) tool [MK99]; we use MiniSAT [ES] for SAT solving. We optimized our implementation to perform incremental SAT encoding using the ability of MiniSAT to solve CNF formulas under a set of unit clause assumptions. We also made ACR return (at each iteration) k counterexamples for the three premises where, k is $|g_1| + |g_2|$.

We compared ACR with learning-based assume guarantee reasoning (based on rule ASYM-AG), on the following examples [PGB+08]: *Gas Station* (3 to 5 customers), *Chiron* – a model of a GUI (2 to 5 event handlers), *Client Server* – a client-server application (6 to 9 clients), and a NASA rover model: *MER* (2 to 4 users competing for two common resources). We used the same two-way decompositions reported in previous experiments. Experiments were performed on a MacBook Pro with a 2.3 GHz Intel Core i7 CPU and with 16 GB RAM running OS X 10.9.4 and a Sun's JDK version 7.

Table 8.1 summarizes our results. For both approaches, we report the model sizes (in states), the analysis time (in seconds) and the assumption sizes (in states). Measuring memory is unreliable due to the garbage collection and the interfacing with MiniSAT via native method calls (our measurements indicate that memory consumption is stable and does not increase dramatically for larger cases). We instead report the maximum numbers of states observed for checking the premises of the two rules. We put a limit of 1800 seconds for each experiment; "–" indicates that the time for that case exceeds this limit.

In all the experiments ACR generates smaller assumptions and in the majority of cases this results in smaller analysis time and state space explored. For larger cases the assumptions generated by ACR are *significantly* smaller. For the Gas Station, ACR significantly outperforms learning in terms of analysis time and states explored, while for all other cases the two approaches are comparable, at smaller sizes. However at larger configurations (Client Server 8 and 9, MER 4) ACR again significantly outperforms the learning-based approach. In all but one case (Chiron 5) the smaller assumptions generated with ACR lead to smaller state spaces for checking the rule premises. Case Chiron 5 is still comparable in terms of running time but it may indicate that the two-way decomposition that we used (found to be optimal for learning in previous studies) may not be optimal for ACR. We plan to investigate this further in future work.

Table 8.1: Comparison of ACR (rule CIRC-AG) and learning (rule ASYM-AG). Best results are shown in bold.

Case	$\|M_1\|$	$\|M_2\|$	ACR Time	$\|g_1\|$	$\|g_2\|$	Premise1	Premise2	Premise3	L^* Time	$\|A\|$	Premise1	Premise2
GasSt 3	1715	643	**26**	**3**	**3**	2588	1093	6	–	>351	>8243	>4045
GasSt 4	14406	1623	**48**	**3**	**3**	19503	2196	4	–	>381	>165836	>47360
GasSt 5	117649	3447	**309**	**3**	**3**	132608	6995	6	–	>207	>560000	>61058
Chiron 2	176	102	1.257	**2**	**2**	134	204	5	**0.5**	9	256	198
Chiron 3	364	1122	**2.013**	**2**	**2**	341	2244	5	2.121	25	492	2736
Chiron 4	703	5559	**3.149**	**2**	**2**	449	6681	5	6.341	45	860	18370
Chiron 5	1905	129228	34	**2**	**2**	1152	258456	5	**33**	122	2101	138537
ClServ 6	729	49	11	**7**	**2**	256	16	10	**8**	256	256	2505
ClServ 7	2187	64	33	**8**	**2**	576	17	10	33	576	576	6455
ClServ 8	6561	81	**53**	**9**	**2**	1280	17	9	138	1280	1280	16199
ClServ 9	19683	100	**249.839**	**10**	**2**	2816	23	14	725	2816	2816	39769
MER 2	225	36	**4.397**	**5**	**2**	30	147	6	4.54	46	313	79
MER 3	1625	76	**35**	**7**	**2**	83	1198	13	50	274	3146	250
MER 4	10625	129	**1220.649**	**9**	**2**	97	7109	9	–	>1210	>128883	>549

Chapter 9

Conclusion and Future Work

We have introduced a novel technique for automatic assume guarantee reasoning using the circular rule CIRC-AG. To the best of our knowledge this is the first work that proposes an automatic compositional verification framework based on a circular rule.

Our algorithm constructs set of joint constraints on the desired assumptions based on counterexamples obtained from checking the premises of the rule, and uses a SAT solver to synthesize minimal assumptions that satisfy the constraints. When $M_1||M_2 \models P$, our algorithm terminates with minimal assumptions that satisfy the premises of the rule CIRC-AG. When $M_1||M_2 \not\models P$ the algorithm returns a counterexample as a witness for the fact that $M_1||M_2$ does not satisfy P.

We have studied the properties of the new algorithm and have experimented with different examples. Our experiments show a significant improvement with respect to learning-based assume guarantee reasoning (based on the rule ASYM-AG) in terms of the sizes of resulting assumptions and in terms of the time consumption.

ACR can be optimized in many ways. Our current implementation checks the three premises of the rule one after the other at each iteration and gets k different counterexamples for each of them. A natural optimization would be to parallelize these checks (e.g. on different machines). We further plan to investigate alphabet refinement and generalization to n-way decomposition (for $n > 2$) – both these techniques significantly enhanced the performance of compositional acyclic techniques [PGB$^+$08]. For the n-way decomposition we can consider a recursive application of our current approach to the system decomposed into two components, each decomposed into two sub-components, etc. Another possibility, which is more involved, would be to directly synthesize n assumptions, one for each component in the system. We leave this for future work. We also plan to explore learning and abstraction-refinement for discovering suitable assumptions. Although these techniques might not guarantee minimal assumptions, they can be less computationally demanding than our current approach.

Bibliography

[AMN05] Rajeev Alur, P. Madhusudan, and Wonhong Nam. Symbolic compositional verification by learning assumptions. In *CAV*, pages 548–562, 2005.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[BPG08] Mihaela Gheorghiu Bobaru, Corina S. Pasareanu, and Dimitra Giannakopoulou. Automated assume-guarantee reasoning by abstraction refinement. In *CAV*, pages 135–148, 2008.

[Bsh95] Nader H. Bshouty. Exact learning boolean function via the monotone theory. *Inf. Comput.*, 123(1):146–153, 1995.

[CCF+10] Yu-Fang Chen, Edmund M. Clarke, Azadeh Farzan, Ming-Hsien Tsai, Yih-Kuen Tsay, and Bow-Yaw Wang. Automated assume-guarantee reasoning through implicit learning. In *CAV*, pages 511–526, 2010.

[CCST05] Sagar Chaki, Edmund M. Clarke, Nishant Sinha, and Prasanna Thati. Automated assume-guarantee reasoning for simulation conformance. In *CAV*, pages 534–547, 2005.

[CFC+09] Yu-Fang Chen, Azadeh Farzan, Edmund M. Clarke, Yih-Kuen Tsay, and Bow-Yaw Wang. Learning minimal separating DFA's for compositional verification. In *TACAS*, pages 31–45, 2009.

[CGJ+03] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT press, December 1999.

[CGP03] Jamieson M. Cobleigh, Dimitra Giannakopoulou, and Corina S. Pasareanu. Learning assumptions for compositional verification. In *TACAS*, pages 331–346, 2003.

[CK99] Shing-Chi Cheung and Jeff Kramer. Checking safety properties using compositional reachability analysis. *ACM Trans. Softw. Eng. Methodol.*, 8(1):49–78, 1999.

[CS07] Sagar Chaki and Ofer Strichman. Optimized l*-based assume-guarantee reasoning. In *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings*, pages 276–291, 2007.

[CW12] Yu-Fang Chen and Bow-Yaw Wang. Learning boolean functions incrementally. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, pages 55–70, 2012.

[dRdBH+00] Willem P. de Roever, Frank S. de Boer, Ulrich Hannemann, Jozef Hooman, Yassine Lakhnech, Mannes Poel, and Job Zwiers. Basic principles of a textbook on the compositional and noncompositional verification of concurrent programs. In *Formale Beschreibungstechniken für verteilte Systeme, 10. GI/ITG-Fachgespräch, Lübeck, Juni 2000*, pages 3–5, 2000.

[ES] Niklas Een and Niklas Ṡorensson. The minisat. http://minisat.se.

[GGP07] Mihaela Gheorghiu, Dimitra Giannakopoulou, and Corina S. Pasareanu. Refining interface alphabets for compositional verification. In *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings*, pages 292–307, 2007.

[GMF08] Anubhav Gupta, Kenneth L. McMillan, and Zhaohui Fu. Automated assumption generation for compositional verification. *Formal Methods in System Design*, 32(3):285–301, 2008.

[GPB05] Dimitra Giannakopoulou, Corina S. Pasareanu, and Howard Barringer. Component verification with automatically generated assumptions. *Autom. Softw. Eng.*, 12(3):297–320, 2005.

[GPQ14] Susanne Graf, Roberto Passerone, and Sophie Quinton. Contract-based reasoning for component systems with rich interactions. In Alberto Sangiovanni-Vincentelli, Haibo Zeng, Marco Di Natale, and Peter Marwedel, editors, *Embedded Systems Development*, volume 20 of *Embedded Systems*, pages 139–154. Springer New York, 2014.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[HQR98] Thomas A. Henzinger, Shaz Qadeer, and Sriram K. Rajamani. You assume, we guarantee: Methodology and case studies. In *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, pages 440–451, 1998.

[HQR00] Thomas A. Henzinger, Shaz Qadeer, and Sriram K. Rajamani. Decomposing refinement proofs using assume-guarantee reasoning. In *Proceedings of the 2000 IEEE/ACM International Conference on Computer-Aided Design, 2000, San Jose, California, USA, November 5-9, 2000*, pages 245–252, 2000.

[LDD+13] Boyang Li, Isil Dillig, Thomas Dillig, Kenneth L. McMillan, and Mooly Sagiv. Synthesis of circular compositional program proofs via abduction. In *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, pages 370–384, 2013.

[Mai03] Patrick Maier. Compositional circular assume-guarantee rules cannot be sound and complete. In *FoSSaCS*, pages 343–357, 2003.

[MC81] Jayadev Misra and K. Mani Chandy. Proofs of networks of processes. *IEEE Trans. Software Eng.*, 7(4):417–426, 1981.

[McM98] Kenneth L. McMillan. Verification of an implementation of Tomasulo's algorithm by compositional model checking. In *CAV*, pages 110–121, 1998.

[McM99a] Kenneth L. McMillan. Circular compositional reasoning about liveness. In *CHARME*, pages 342–345, 1999.

[McM99b] Kenneth L. McMillan. Verification of infinite state systems by compositional model checking. In *CHARME*, pages 219–234, 1999.

[MK99] J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. John Wiley & Sons, 1999.

[NT00] Kedar S. Namjoshi and Richard J. Trefler. On the competeness of compositional reasoning. In *CAV*, pages 139–153, 2000.

[PG06] Corina S. Pasareanu and Dimitra Giannakopoulou. Towards a compositional spin. In *SPIN*, pages 234–251, 2006.

[PGB+08] Corina S. Pasareanu, Dimitra Giannakopoulou, Mihaela Gheorghiu Bobaru, Jamieson M. Cobleigh, and Howard Barringer. Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. *Formal Methods in System Design*, 32(3):175–205, 2008.

[Pnu85] Amir Pnueli. In transition from global to modular temporal reasoning about programs. In *Logics and Models of Concurrent Systems, NATO ASI Series*, 1985.

[Rus01] John Rushby. Formal verification of mcmillan's compositional assume-guarantee rule. In *CSL Technical Report, SRI*, 2001.

[TB97] Serdar Tasiran and Robert K. Brayton. STARI: A case study in compositional and hierarchical timing verification. In *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, pages 191–201, 1997.