Logic-Based Analysis of Reactive Systems: Hiding, Composition, and Abstraction

Stephan Merz

Institut für Informatik, Ludwig-Maximilians-Universität München
Oettingenstr. 67, D-80538 München, Germany

Habilitationsschrift

München, im Dezember 2001

Abstract. We study aspects of three central issues in the formal design of reactive systems: *hiding* refers to the ability to control the visibility of specification detail, *compositionality* enables large systems to be assembled from separate components, and *abstraction* promises a higher degree of automation in the verification of large systems. The work is cast in the framework of temporal logic, and proof rules are provided to support all three concepts.

Acknowledgements

I am very grateful to Fred Kröger and Martin Wirsing for providing me with a scientific home and for their support over many years. I would also like to express my special gratitude to Leslie Lamport for providing me with a topic to write about, for hosting me at the Systems Research Center, and for many hours of discussion. His clear appreciation of problems and non-problems have been a constant example. I am particularly grateful that he has been willing to act as a referee for this thesis. I have been fortunate to work with Martín Abadi who has also influenced my work. Manfred Broy is acknowledged for admitting me to work in his group and for interesting discussions.

Many friends and colleagues have provided stimulus and support in various ways, among them Alexander^2, Bernhard, Birgit, Dirk, Dominique^2, François, Frank, Heni, Holger, Júlia, Olaf, Rolf, and Toni. Thank you all for good friendship and for interesting discussions over the lunch table and in the seminar room.

I cannot even begin to express my gratitude towards Beate for her love and for her unfailing encouragement and support. Without her, this thesis would never have been written. Our children Agnes, Bernhard, and Roland have had to see their father disappear behind his desk on many a weekend; Beate has taught them to accept this behavior as a given.

Contents

Chapter 1

Introduction

1.1 Background

Computerized systems have become ubiquitous in our everyday lives. We rely on digital controllers to supervise critical functions of cars, airplanes, medical equipment, and industrial plants. Digital switching technology has replaced analog components in the telecommunication industry, and security protocols enable e-commerce applications and privacy. Where systems can jeopardize human lives or health, we speak of *safety-critical systems*. Unlike standard desktop programs whose users have mostly become accustomed to occasional malfunction, errors in safety-critical systems are unacceptable [20, 65]. Industry concerns and government regulations require certain levels of assurance against errors in such systems. Similar requirements about the quality of software systems are also imposed when failures lead to economically significant loss of money. Well-publicized instances of such errors were the so-called FDIV bug in early Intel Pentium processors and the loss of the Ariane-5 launcher.

One of the most challenging problems facing today's software engineers and computer scientists is therefore to find ways and establish techniques in order to reduce the number of errors in safety-critical systems. This can only be successfully attempted when the design process that underlies the production of such systems is understood. It is often estimated that some 80–90% of the overall development cost of a typical embedded system is due to the design of the control software. It proceeds in many stages where a multitude of documents is produced and many designers need to interact and understand the overall design, its decomposition into components and their interfaces. The design documents have to unambiguously describe the function of components at an adequate level of abstraction. For example, it is hardly possible to precisely describe the protocol required at some interface in natural language because such descriptions are open to different interpretations and are hard to check for omissions. At the other extreme, a machine code description of the same protocol would contain complete detail and be unambiguous, but it would be utterly incomprehensible.

The research in *formal methods* attempts to provide languages that allow systems to be described (or *specified*) at adequate levels of abstraction and support various forms of analysis, from static and type checking to prototyping and formal proof, to be performed on specifications. A description is *formal* if it is built from a well-defined alphabet of elementary symbols following well-defined syntactic rules and if it has a formally (i.e., mathematically) defined meaning. The main advantage of a formal description over an informal one is that it lends itself to formal analysis. For example, it can be checked whether the description is free of contradictions or whether it implies certain desired properties. Formal methods support the design process of systems over succesive stages by an appropriate *refinement* notion; proof conditions ensure that a lower-level, more detailed specification correctly implements a higher-level one.

This thesis is about central concerns regarding the temporal logic-based specification and analysis of safety-critical *reactive systems* [42, 70]. Traditional models of computation emphasize the process of computing some output as a function of given input values; these are inadequate for the description of the kind of systems that we have mentioned above, which are characterized by maintaining an ongoing interaction with their environment. The behavior of reactive systems is represented by sequences of states or configurations. The models that are used to describe reactive systems are usually based on transition systems: they describe the set of possible (even erroneous!) system states, a subset of initial states, and (non-deterministic) rules for moving from one state to its successor state. Another important ingredient in the description of transition systems for modelling reactive systems are fairness and liveness conditions that require some actions to be eventually taken or some state to be eventually reached. These conditions help to balance the local non-determinism of transition systems of choosing among actions that are permitted at a given source state and lead to different possible target states.

The use of temporal logics for the specification and verification of reactive systems goes back to Pnueli's seminal paper on temporal logic [84] and is the topic of many textbooks [57, 70, 71, 94]. Formulas of temporal logic are interpreted over runs of transition systems and can thus express properties of reactive systems. The main advantage of temporal logics over other formalisms that are directly based on transition systems, such as I/O automata [66] or B [9], is that fairness and liveness properties can be succinctly and abstractly expressed in temporal logic without resorting to low-level constructs such as counters and queues. Similarly, temporal logic provides proof rules for verifying that a transition system implements liveness properties. However, the traditional, *property-oriented* use of temporal logic where a specification consists of a list of desired correctness properties does not scale to realistic systems. First, it is very difficult to verify that such a description is consistent. Second, it becomes unmanageable to convince oneself and others that no important requirements have been omitted, that is, that the specification completely describes the desired system behavior. Academic case studies such as the RPC-Memory problem [23] have shown that even expert users of temporal logic are not capable of writing a correct specification of small systems in this style.

In contrast, it is much easier to unambiguously specify the behavior of a state machine. Moreover, it is usually relatively straightforward to check for the consistency of such a specification. In particular, if a state machine specification is *machine closed* [2], it is enough to verify that the initial condition is satisfiable and that every (reachable) state has at least one possible successor. This observation makes the so-called *model-oriented* style of specification very attractive. The main potential drawback of this style is the danger of over-specification: later design stages should not be bound to the particular representation of the state machine present in the higher-level specification.

Lamport's Temporal Logic of Actions [62] includes, in a sense, the best of these two worlds: TLA specifications combine the description of a state machine for specifying the initial states and the next-state relation with temporal logic formulas for writing fairness and liveness properties. Over-specification is avoided by *hiding* the internal state representation, expressed by existential quantification over internal state components. Therefore, later implementations are free to choose some more appropriate representation that produces the same (possibly more detailed) externally visible behavior.

An important feature of TLA that supports refinement is that its formulas are *invariant with respect to stuttering*, that is, finite repetitions of indistinguishable states. The idea is that refinements are free to add further detail that was not present in the high-level description. In particular, state transitions at the abstract level may be refined into a series of transitions of the implementation; only the final transition changes the part of the state that is visible at the higher level. Behaviors of the implementation can therefore be viewed as behaviors of the specification, with additional stuttering steps inserted. Because the specification is insensitive to these stuttering steps, the correctness of the refinement can be expressed as trace inclusion. Equivalently, specification C implements specification A if and only if the implication $C \rightarrow A$ is valid. Similarly, the (parallel) composition of two components specified by A and B can be represented by the conjunction $A \wedge B$ of the component specifications. Because there is no formal distinction between properties and specifications, and because structural notions such as composition, hiding, and refinement are expressed by logical connectives, a single collection of TLA proof rules suffices to establish correctness properties and refinement.

1.2 Overview and main results

This thesis attempts to address some perceived shortcomings and further improve the applicability of the TLA approach, within the same general framework. We start from the observation that the language of TLA is well-suited to express system specifications but less so for higher-level requirement specifications. In particular, while state-based properties are expressed much as in standard temporal logics, action-based properties are not expressed so easily. This is due to the definition of the syntax of TLA, which treats action formulas as "second-class citizens": whereas action formulas can occur as subformulas of temporal formulas, they may not themselves contain temporal operators. On a more theoretical note, whereas Lamport gives a relatively complete set of proof rules for system verification, no complete axiomatization in the traditional sense exists in the literature, not even for the propositional fragment. (Abadi [1] defined such a proof system for a previous version of TLA, but it cannot be adapted to the present version.) In chapter 2 we define a generalization TLA* of TLA that is symmetric in the definition of state-based and action-based properties and give a complete axiomatization of the propositional fragment of TLA*. We also show that TLA* formulas are again invariant under stuttering. Our proofs indicate that the lack of an axiomatization of TLA may in fact be related to the restricted expressiveness; in particular, it is impossible to express an adequate induction axiom in (unquantified) TLA.

Quantified TLA* is the object of study of chapter 4; as we have seen, quantification over state variables is essential in model-oriented formalisms to avoid overspecification. We first restrict ourselves to the propositional fragment of TLA*, which is easier to analyze and establishes a frame of reference for the full logic. We show that the quantified versions of TLA and TLA* are equi-expressive. The main part of the chapter is devoted to proving a completeness theorem for quantified propositional TLA*. Intuitively, proving refinement of a specification with hidden state components requires to reconstruct the internal state of the original specification from that of the implementation. However, if the internal state of the implementation is significantly different from that of the specification it may be impossible to define such a mapping state by state. It has been proposed [2, 67] to solve this problem by introducing (forward or backward) inductive definitions. We define corresponding introduction rules for auxiliary variables in the language of TLA* that are used in the completeness proof. The completeness proof itself relies on an encoding of finite automata on infinite words in TLA* (a similar proof has been sketched for quantified propositional linear-time temporal logic in [54]). Because classical ω-automata can not be used immediately as they would be sensitive to stuttering, chapter 3 develops a variant of the standard theory of Büchi automata [98, 99] for ω-languages that are closed under stuttering equivalence. As a corollary to the completeness proof, we obtain decidability of quantified propositional TLA*. Finally, we introduce a first-order version of TLA* towards the end of chapter 4 and discuss the necessary modifications and extensions of proof rules.

Modularization is a key technique that enables the design of large systems by dividing such systems into separate components with clear interfaces. Individual components are rarely expected to operate in arbitrary environments, leading to the development of techniques for the specification of *open systems* that commit to proper functioning only when assuming the environment to be well-behaved. A large body of literature discusses how to express open-system specifications in such a way as to obtain useful rules for the verification of systems assembled from components. In particular, Abadi and Lamport [4] have proposed such rules for TLA specifications. In chapter 5, representing joint work with M. Abadi [6], we justify their work from a more primitive, intuitionistic logic of specifications. We show that similar composition rules for different formalisms, such as CTL*, can be justified from the same basic rule in the abstract logic.

Chapters 6 and 7 are more oriented towards practical verification, which is dominated by the two paradigms of state space exploration (exemplified by the model checking technique) and deductive verification. Although mere system specification can already be beneficial in design in that designers are required to invent adequate abstractions and are rewarded with unambiguous documentation, the stringent use of formal methods requires support by powerful and usable verification machinery. Model checking [29, 45, 75] has gained wide acceptance because it is fully automatic and because it produces useful debugging information in case of errors, but it is intrinsically limited to a restricted class of (usually finite-state) systems. Theorem proving [21, 82, 83] can in principle be applied to arbitrary systems, but it requires a high degree of expertise, and the failure to prove a theorem does not necessarily indicate an error in the design. A recent subject of research has therefore been to find ways how to combine the advantages of model checking and theorem proving. *Abstraction* has been accepted to be one of the most promising techniques for such a combination [53]. Basically, the idea consists in reducing the problem of verifying some property P of a large (perhaps infinite-state) system Φ to one of verifying some property $\overline{P}$ of a small, finite-state system $\overline{\Phi}$. If the verification problem succeeds at the abstract level, the concrete system Φ is guaranteed to satisfy P. On the other hand, a failure of the abstract verification problem need not indicate a design error, but the resulting counter-example can be used to refine the abstraction. In chapter 6 we recast the basic framework of homomorphic abstraction in the form of TLA* proof rules. We then concentrate on techniques that allow liveness properties to be inferred of an abstract model: this is usually the most tedious part of deductive verification, and being able to apply model checking can significantly reduce the amount of user interaction required.

In chapter 7 we study a particular format of abstractions that we have developed jointly with D. Cansell and D. Méry, based on the framework of Boolean abstractions advocated, among others, by [19, 40, 68]. One of the attractive features of the proposed format of predicate diagrams is the fact that it provides a visual, state machine-like, representation of system abstractions. We state proof obligations that can be used to verify that a predicate diagram is a correct abstraction of a system specification and present an encoding of predicate diagrams as finite transition systems that enables properties to be verified by model checking. We also discuss how abstract evaluation techniques can be applied to semi-automatically generate predicate diagrams. Finally, we introduce a refinement notion at the level of predicate diagrams. The techniques are illustrated by a number of examples. In particular, we show that certain infinite-state systems can in this way be verified without any user intervention.

Chapter 2

A Generalized Temporal Logic of Actions

2.1 Overview

We now define the logic that we use to describe our methods for representing and reasoning about reactive systems, and study adequate proof rules. Our logic is a variant of linear-time temporal logic, inspired by Lamport's Temporal Logic of Actions [62]. As in TLA, we represent system specifications as well as their properties by formulas. This allows us to use logical connectives such as conjunction and implication to express relationships between specifications or between specifications and properties such as composition and refinement. As observed by Lamport [61], this requires formulas to be insensitive to finite "stuttering", that is, repetitions of states that are indistinguishable at the current level of abstraction (and in particular, grain of atomicity). Our logic differs from TLA in that it is based on a symmetrical, mutually recursive definition of (stuttering sensitive) pre-formulas and (stuttering invariant) formulas rather than a layer of formulas on top of a layer of nontemporal action formulas as in TLA. Although more laborious at first sight, this generalized notion of formulas allows certain high-level requirements to be expressed more naturally. Besides, it simplifies the theory; in particular, it allows for a relatively straightforward axiomatization.

2.2 Propositional TLA*

We define a propositional version of TLA* (pTLA*) and prove that its formulas are invariant under stuttering. A first-order extension will be considered in section 4.7.

2.2.1 Syntax and semantics

Assume given a denumerable set $\mathcal{V}$ of atomic propositions.

Definition 2.1 (syntax) *Formulas and pre-formulas of* pTLA* *are inductively defined as follows.*

1. **false** *is a formula, as is every atomic proposition* $v \in \mathcal{V}$.
2. *If* F, G *are formulas then* $F \to G$, *and* $\Box F$ *are formulas.*
3. *If* A *is a pre-formula and* $v \in \mathcal{V}$ *then* $\Box[A]_v$ *is a formula.*
4. *If* F *is a formula then* F *and* $\circ F$ *are pre-formulas.*
5. *If* A, B *are pre-formulas then* $A \to B$ *is a pre-formula.*

For a (pre-)formula A*, we denote by* $At(A)$ *the set of atomic propositions that occur in* A.

We will use symbols such as F, G for formulas and A, B for pre-formulas. Note that, as in TLA, $\Box[\_]_v$ is considered to be a separate operator for all $v \in \mathcal{V}$ different from $\Box$, and that $[A]_v$ is not a formula.

When writing formulas, we use many derived connectives: standard propositional operators include **true**, $\neg$, $\wedge$, $\vee$, and $\leftrightarrow$ for both formulas and pre-formulas. We sometimes write v' instead of $\circ v$ when $v \in \mathcal{V}$ is an atomic proposition. For (possibly primed) atomic propositions we sometimes use the equality symbol to denote equivalence, and write, for example, $w' = v$ instead of $w' \leftrightarrow v$. If $V = \{v_1, \ldots, v_n\} \subseteq \mathcal{V}$ is a finite set of atomic propositions, we write $V' = V$ for the pre-formula

$$v_1' = v_1 \wedge \ldots \wedge v_n' = v_n$$

and $\Box[A]_V$ or $\Box[A]_{v_1,\ldots,v_n}$ to denote the formula

$$\Box[A]_{v_1} \wedge \ldots \wedge \Box[A]_{v_n}$$

In particular, $\Box[A]_\emptyset$ equals **true**. Stretching notation even further, we write $\Box[A]_F$ for the formula[1] $\Box[A \vee (\circ F \leftrightarrow F)]_{At(F)}$. We write $\Diamond F$ for the formula $\neg\Box\neg F$ and $\Diamond\langle A\rangle_v$ for $\neg\Box[\neg A]_v$. Consequently, $\Diamond\langle A\rangle_{v_1,\ldots,v_n}$ denotes

$$\Diamond\langle A\rangle_{v_1} \vee \ldots \vee \Diamond\langle A\rangle_{v_n}$$

[1] This notation introduces an ambiguity when $F \equiv v$ is an atomic proposition. However, both possible readings are equivalent with respect to the semantics of definition 2.2 below.

and $\Diamond\langle A\rangle_F$ abbreviates $\Diamond\langle A \wedge \neg(\circ F \leftrightarrow F)\rangle_{At(F)}$. Finally, we use the notation $[A]_F$ and $\langle A\rangle_F$ for the pre-formulas $A \vee (\circ F \leftrightarrow F)$ and $A \wedge \neg(\circ F \leftrightarrow F)$, respectively.

Formulas of pTLA* are similar to those of propositional TLA. For example, the formula $\Box[A]_v$ asserts that A holds whenever v changes value. It is, however, more general in that A may itself contain temporal operators whereas TLA would allow only action formulas, i.e. Boolean combinations of primed and unprimed atomic propositions, to occur inside brackets. Our more liberal syntax allows high-level requirements to be expressed more easily. As an example, consider a specification that asserts that whenever an $\langle A\rangle_v$ action has occurred, only B actions are allowed to change the value of w. In pTLA*, this requirement can be expressed as the formula

$$\Box\Big[A \rightarrow \Box[B]_w\Big]_v \tag{2.1}$$

just as the corresponding assertion for formulas F and G would be expressed as

$$\Box(F \rightarrow \Box G)$$

In TLA, the latter is a formula, while the former is not. A more careful comparison with standard TLA appears in section 4.3. In particular, we will see that (2.1) is expressible in quantified TLA.

We now turn to the semantics of pTLA*. A *state* is a boolean valuation $s : \mathcal{V} \rightarrow \{\mathsf{tt}, \mathsf{ff}\}$ of the atomic propositions. A *behavior* $\sigma = s_0 s_1 \ldots$ is an infinite sequence of states. We now define what it means for a pre-formula to hold of a behavior σ, written $\sigma \mathrel{|\!\approx} A$. Because every formula is also a pre-formula, this also defines the semantics of formulas.

Definition 2.2 *The semantics of pre-formulas is given by the relation $\mathrel{|\!\approx}$, which is inductively defined as follows:*

$\sigma \mathrel{|\!\not\approx}$ **false**.

$\sigma \mathrel{|\!\approx} v$	*iff*	$s_0(v) = \mathsf{tt}$ *(for $v \in \mathcal{V}$).*
$\sigma \mathrel{|\!\approx} A \rightarrow B$	*iff*	$\sigma \mathrel{|\!\approx} A$ *implies* $\sigma \mathrel{|\!\approx} B$.
$\sigma \mathrel{|\!\approx} \Box F$	*iff*	$\sigma[i..] \mathrel{|\!\approx} F$ *holds for all* $i \in \mathbb{N}$.
$\sigma \mathrel{|\!\approx} \Box[A]_v$	*iff*	*for all* $i \in \mathbb{N}$, $s_i(v) = s_{i+1}(v)$ *or* $\sigma[i..] \mathrel{|\!\approx} A$.
$\sigma \mathrel{|\!\approx} \circ F$	*iff*	$\sigma[1..] \mathrel{|\!\approx} F$.

For a formula F, we usually write $\sigma \models F$ instead of $\sigma \mathrel{|\!\approx} F$.

A behavior σ such that $\sigma \mathrel{|\!\approx} A$ holds is called a *model* of A. We say that a formula F is *valid over a behavior* σ iff $\sigma[n..] \models F$ holds for all $n \in \mathbb{N}$. Formula F

follows from a set $\mathcal{F}$ of formulas (written $\mathcal{F} \models F$) iff F is valid over all behaviors over which all formulas $G \in \mathcal{F}$ are valid. Finally, F is *valid* (written $\models F$) iff it is valid over all behaviors, which is equivalent to saying that it follows from $\emptyset$.

This so-called *floating validity* is the traditional definition for modal logics, rather than the alternative *anchored validity* used by Lamport [62] uses. It is well known that either choice defines the same set of valid formulas, although the consequence relation is different. We prefer floating validity because it is easier to axiomatize.

We say that a (pre-)formula is tautological if it results from a propositional tautology A of classical logic by consistently replacing atomic subformulas of A by formulas or pre-formulas. It is easy to see that every tautological formula is valid.

2.2.2 Stuttering invariance

Definition 2.1 allows the $\Box$ operator to be applied only to formulas. For example, $\Box \circ v$ is not a pre-formula, although $\circ \Box v$ is. This restriction is designed to ensure invariance under stuttering: consider, for example, $\Box[\Box(p \rightarrow \circ q)]_p$, which is not a pTLA* formula, and the behaviors σ and τ, where τ differs from σ in the repetition of the second state, as illustrated by the following diagram (where $\sim$ indicates "don't care"):

$$
\begin{array}{lcccc}
\sigma: & \neg p & p & \sim & \sim \\
 & \sim & \neg q & q & q
\end{array}
\qquad
\begin{array}{lccccc}
\tau: & \neg p & p & p & \sim & \sim \\
 & \sim & \neg q & \neg q & q & q
\end{array}
$$

Assuming the last state to repeat indefinitely, $\Box[\Box(p \rightarrow \circ q)]_p$ clearly holds of σ (w.r.t. to the canonical extension of definition 2.2), but not of τ.

We now formally define stuttering equivalence and prove that pTLA* formulas do not distinguish between stuttering equivalent behaviors.

Definition 2.3 (stuttering equivalence) *Let $V \subseteq \mathcal{V}$.*

1. *Two states s, t are called* V-similar, *written $s =_V t$ iff $s(v) = t(v)$ for all $v \in V$. Two behaviors σ, τ are called V-similar iff $s_i =_V t_i$ holds for all i.*

2. V-stuttering equivalence, *written $\simeq_V$, is the smallest equivalence relation on behaviors that identifies $\rho \circ \langle s \rangle \circ \sigma$ and $\rho \circ \langle tu \rangle \circ \sigma$, for any finite sequence of states ρ, infinite sequence of states σ, and pairwise V-similar states s, t, u.*

3. *Stuttering equivalence (written $\simeq$) is $\mathcal{V}$-stuttering equivalence.*

It follows that $\sigma \simeq_V \tau$ implies $\sigma \simeq_W \tau$ whenever $W \subseteq V$ holds. In particular, stuttering equivalence is the finest relation among all $\simeq_V$. Note also that two states s, t are $\mathcal{V}$-similar iff they are equal. Let us list some elementary facts about stuttering equivalent behaviors.

Proposition 2.4 *Assume that $\sigma = s_0 s_1 \ldots$ and $\tau = t_0 t_1 \ldots$ are behaviors such that $\sigma \simeq_V \tau$ holds for some $V \subseteq \mathcal{V}$.*

1. $t_0 =_V s_0$.
2. *For every $n \in \mathbb{N}$ there is some $m \in \mathbb{N}$ such that both $\sigma[n\,..] \simeq_V \tau[m\,..]$ and $\sigma[n+1\,..] \simeq_V \tau[m+1\,..]$.*

Replacing maximal finite repetitions of V-similar states by a single occurrence of, say, the first of these states, it follows that for any behavior σ there exists a V-stuttering equivalent behavior $\natural_V\sigma = s_0 s_1 \ldots$ that contains successive V-similar states only if it ends in V-stuttering, that is, for all $i \in \mathbb{N}$, $s_i =_V s_{i+1}$ holds only if $s_i =_V s_j$ for all $j \geq i$. We call any such behavior $\natural_V\sigma$ a *V-stuttering free variant* of σ. Two behaviors σ, τ are V-stuttering equivalent if and only if $\natural_V\sigma =_V \natural_V\tau$ holds, where $\natural_V\sigma$ and $\natural_V\tau$ denote arbitrary V-stuttering free variants of σ and τ.

Theorem 2.5 (stuttering invariance) *Assume that F is a formula, A is a pre-formula, and that σ, τ are behaviors.*

1. *If $\sigma \simeq_{At(F)} \tau$ then $\sigma \models F$ iff $\tau \models F$.*
2. *If $\sigma \simeq_{At(A)} \tau$ and $\sigma[1\,..] \simeq_{At(A)} \tau[1\,..]$ then $\sigma \mathrel{|\!\approx} A$ iff $\tau \mathrel{|\!\approx} A$.*

Proof. We simultaneously prove the assertions by induction on the structure of formulas and pre-formulas. We first consider the different cases in the definition of formulas F.

false : trivial.

$F \in \mathcal{V}$: The assertion follows from proposition 2.4.1, since $s_0(F) = t_0(F)$.

$F \to G$: By assumption we have $\sigma \simeq_{At(F \to G)} \tau$; moreover, $At(F) \subseteq At(F \to G)$ and $At(G) \subseteq At(F \to G)$. It therefore follows that both $\sigma \simeq_{At(F)} \tau$ and $\sigma \simeq_{At(G)} \tau$. This observation, together with the induction hypothesis, proves the assertion.

$\Box F$: By symmetry of $\simeq_{At(\Box F)}$, it is enough to prove "if". So assume that $\tau \models \Box F$, and let $n \in \mathbb{N}$ be arbitrary. Proposition 2.4.2 implies that there exists some $m \in \mathbb{N}$ such that $\sigma[n\,..] \simeq_{At(\Box F)} \tau[m\,..]$. From $\tau \models \Box F$ we conclude $\tau[m\,..] \models F$, and therefore $\sigma[n\,..] \models F$ by induction hypothesis, since $At(\Box F) = At(F)$.

(ax0)	$\vdash F \quad$ for F tautological	(pax0)	$\mid\!\sim A \quad$ for A tautological
(ax1)	$\vdash \Box F \rightarrow F$	(pax1)	$\mid\!\sim \circ\neg A \leftrightarrow \neg\circ A$
(ax2)	$\vdash \Box F \rightarrow \Box[\Box F]_v$	(pax2)	$\mid\!\sim \circ(F \rightarrow G) \rightarrow (\circ F \rightarrow \circ G)$
(ax3)	$\vdash \Box[F \rightarrow \circ F]_F \rightarrow (F \rightarrow \Box F)$	(pax3)	$\mid\!\sim \Box F \rightarrow \circ\Box F$
(ax4)	$\vdash \Box[A \rightarrow B]_v \rightarrow (\Box[A]_v \rightarrow \Box[B]_v)$	(pax4)	$\mid\!\sim \Box[A]_v \leftrightarrow [A]_v \wedge \circ\Box[A]_v$
(ax5)	$\vdash \Box[v' \neq v]_v$	(pax5)	$\mid\!\sim \circ\Box F \rightarrow \Box[\circ F]_v$

(mp) $\dfrac{\vdash F \qquad \vdash F \rightarrow G}{\vdash G}$ (pmp) $\dfrac{\mid\!\sim A \qquad \mid\!\sim A \rightarrow B}{\mid\!\sim B}$

(sq) $\dfrac{\mid\!\sim A}{\vdash \Box[A]_v}$ (pre) $\dfrac{\vdash F}{\mid\!\sim F}$ (nex) $\dfrac{\vdash F}{\mid\!\sim \circ F}$

Figure 2.1: The proof system Σ_{TLA^*}.

$\Box[A]_v$: Again, we need only prove the "if" part. Assume that $\tau \models \Box[A]_v$, and let $n \in \mathbb{N}$ be arbitrary. Proposition 2.4.2 ensures the existence of some $m \in \mathbb{N}$ such that $\sigma[n..] \simeq_{At(\Box[A]_v)} \tau[m..]$ and also $\sigma[n+1..] \simeq_{At(\Box[A]_v)} \tau[m+1..]$; choose some such m. Proposition 2.4.1 implies that $s_n(v) = t_m(v)$ and $s_{n+1}(v) = t_{m+1}(v)$. If $t_m(v) = t_{m+1}(v)$, it follows that $s_n(v) = s_{n+1}(v)$, and we are done. Otherwise, by the assumption $\tau \models \Box[A]_v$ it follows that $\tau[m..] \mid\!\approx A$, and the induction hypothesis (for assertion 2) gives $\sigma[n..] \mid\!\approx A$ because $\simeq_{At(\Box[A]_v)} \subseteq \simeq_{At(A)}$.

Turning to assertion 2, we consider the cases in the definition of pre-formulas:

A a formula : immediate from the induction hypothesis for assertion 1.

$\circ F$: The assumption that $\sigma[1..] \simeq_{At(\circ F)} \tau[1..]$ and the induction hypothesis for assertion 1 imply $\sigma[1..] \models F$ iff $\tau[1..] \models F$, and therefore $\sigma \mid\!\approx \circ F$ iff $\tau \mid\!\approx \circ F$.

$A \rightarrow B$: analogous to the proof for formulas. Q.E.D.

2.3 An Axiomatization of pTLA*

We now present a proof system Σ_{TLA^*} for pTLA* and prove its adequacy. Σ_{TLA^*} is based on two provability relations $\vdash$ and $\mid\!\sim$ for formulas and pre-formulas. Alternatively, we could have defined a single provability relation that applied to both formulas and pre-formulas, as in the definition of the semantics of pTLA*. This would amount to adding a rule to Σ_{TLA^*} that allowed $\vdash F$ to be inferred from $\mid\!\sim F$, so the formulation with two relations is stronger. In fact, the admissibility of this additional rule will be shown as theorem 2.8 below. An alternative "homogeneous" proof system based on a single relation that applies only to formulas will be considered in section 2.5. Figure 2.1 contains the axioms and rules that define $\vdash$ and $|\!\!\sim$. We extend $\vdash$ to a relation between sets of formulas and formulas by defining $\mathcal{F} \vdash F$ iff $\vdash F$ can be established from the axioms and rules of Σ_{TLA^*} if additionally $\vdash G$ is assumed for all formulas $G \in \mathcal{F}$, and similarly define $\mathcal{F} \mathrel{|\!\!\sim} A$. Because we are ultimately only interested in the relation $\vdash$ for formulas, we do not allow pre-formulas to occur in the set $\mathcal{F}$ of hypotheses.

Many of the axioms and rules of Σ_{TLA^*} are familiar from propositional linear-time temporal logic [39, 57]. First observe that both $\vdash$ and $|\!\!\sim$ contain a classical propositional calculus. Axiom (ax3) is a formulation of the induction axiom, which could not be stated in full generality in standard TLA. Axiom (ax5) asserts that the pre-formula A in $\Box[A]_v$ is evaluated only when v changes value. Axiom (pax1) expresses that time is linear. We cannot state an induction principle for formulas of the form $\Box[A]_v$ because $\circ A$ or $\circ[A]_v$ are not even pre-formulas. Therefore, we can only state a fixed-point property of $\Box[A]_v$ in axiom (pax4). Axiom (pax5) asserts a form of commutativity for the $\circ$ and $\Box$ operators. The rules (sq) and (nex) reflect the floating definition of validity. The necessitation rule

$$\text{(alw)} \qquad \frac{\vdash F}{\vdash \Box F}$$

can be derived in Σ_{TLA^*}, cf. section A.1 in the appendix for a derivation of this and some other laws in Σ_{TLA^*}. Versions of (ax2), (ax4), (pax4), (pax5) and rule (sq) where the subscript v is replaced by a finite set V or by a formula F are also easily derived in Σ_{TLA^*}.

Theorem 2.6 (Soundness) *For any set $\mathcal{F}$ of formulas, $\mathcal{F} \vdash F$ implies $\mathcal{F} \models F$.*

Proof. By induction of the assumed derivation, we show that whenever either $\mathcal{F} \vdash F$ or $\mathcal{F} \mathrel{|\!\!\sim} A$ holds then for any behavior σ such that all formulas in $\mathcal{F}$ are valid over σ then so are F or A, respectively. We only give a few characteristic cases:

(ax3) It suffices to prove $\sigma \models \Box[F \rightarrow \circ F]_F \rightarrow (F \rightarrow \Box F)$, for any formula F and any behavior $\sigma = s_0 s_1 \ldots$. So suppose $\sigma \models \Box[F \rightarrow \circ F]_F$ and $\sigma \models F$. We prove $\sigma[n..] \models F$ for all $n \in \mathbb{N}$, by induction on n. The base case being trivial, assume that $\sigma[n..] \models F$. If $s_n \simeq_{At(F)} s_{n+1}$, we have $\sigma[n..] \simeq_{At(F)} \sigma[n+1..]$, and theorem 2.5 ensures that $\sigma[n+1..] \models F$. Otherwise, there is some $v \in At(F)$ such that $s_n(v) \neq s_{n+1}(v)$, and the assumption $\sigma \models \Box[F \rightarrow \circ F]_F$ implies that $\sigma[n..] \mathrel{|\!\!\approx} (F \rightarrow \circ F) \vee (\circ F \leftrightarrow F)$, and again it follows that $\sigma[n+1..] \models F$.

(pax5) Suppose $\sigma \mathrel{|\!\!\approx} \circ\Box F$. We prove that $\sigma \models \Box[\circ F]_v$. Let $m \in \mathbb{N}$ be arbitrary. The assumption ensures that $\sigma[m+1..] \models F$, and therefore $\sigma[m..] \mathrel{|\!\!\approx} \circ F$. This suffices.

(sq) Assume that $\mathcal{F} \mid\!\sim A$, that σ is some behavior such that all formulas in $\mathcal{F}$ are valid over σ, and that $n \in \mathbb{N}$. We need to prove that $\sigma[n..] \models \Box[A]_v$. So let $m \in \mathbb{N}$ be arbitrary. By induction hypothesis, we know that $\sigma[n+m..] \mid\!\approx A$, and therefore $(\sigma[n..])[m..] \mid\!\approx A$. This suffices. Q.E.D.

We also have a deduction theorem for Σ_{TLA^*}.

Theorem 2.7 *Let $\mathcal{F}$ be a set of formulas, F, G be formulas, and A be a preformula.*

(i) $\mathcal{F} \cup \{F\} \vdash G$ *iff* $\mathcal{F} \vdash \Box F \to G$ (ii) $\mathcal{F} \cup \{F\} \mid\!\sim A$ *iff* $\mathcal{F} \mid\!\sim \Box F \to A$

Proof. "if": Assume $\mathcal{F} \vdash \Box F \to G$. A fortiori, we have $\mathcal{F} \cup \{F\} \vdash \Box F \to G$. The derived rule (alw) implies that $\mathcal{F} \cup \{F\} \vdash \Box F$, and therefore we have $\mathcal{F} \cup \{F\} \vdash G$ by (mp). The second assertion is proven similarly.

"only if": The proof is by induction on the assumed derivations of $\mathcal{F} \cup \{F\} \vdash G$ and $\mathcal{F} \cup \{F\} \mid\!\sim A$ (simultaneously for all F and A).

- If G is an axiom or $G \in \mathcal{F}$, we have $\mathcal{F} \vdash G$, and $\mathcal{F} \vdash \Box F \to G$ follows by propositional reasoning. The same argument applies for the second assertion when A is an axiom.
- If G is F, then $\mathcal{F} \vdash \Box F \to F$ is an instance of (ax1).
- If G results from an application of (mp) to previously derived formulas $H \to G$ and H, then the induction hypothesis implies $\mathcal{F} \vdash \Box F \to (H \to G)$ as well as $\mathcal{F} \vdash \Box F \to H$, from which we conclude $\mathcal{F} \vdash \Box F \to G$ by propositional reasoning. The same argument holds for (pmp).
- Assume that G results from an application of (sq), say, $G \equiv \Box[B]_v$. By induction hypothesis, we have $\mathcal{F} \mid\!\sim \Box F \to B$, and we continue as follows:

(1)	$\mid\!\sim \Box F \to B$	(ind.hyp.)
(2)	$\vdash \Box[\Box F \to B]_v$	(sq)(1)
(3)	$\vdash \Box[\Box F \to B]_v \to (\Box[\Box F]_v \to \Box[B]_v)$	(ax4)
(4)	$\vdash \Box F \to \Box[\Box F]_v$	(ax2)
(5)	$\vdash \Box F \to \Box[B]_v$	(prop)(2)(3)(4)

- If G results from an application of (pre), then by induction hypothesis we have $\mathcal{F} \vdash \Box F \to G$, and therefore also $\mathcal{F} \mid\!\sim \Box F \to G$, by (pre).
- Assume that $G \equiv \circ H$ results from an application of (nex). By induction hypothesis, we have $\mathcal{F} \vdash \Box F \to H$, and we continue as follows:

(1)	$\vdash \Box F \to H$	(ind.hyp.)
(2)	$\mid\!\sim \circ(\Box F \to H)$	(nex)(1)
(3)	$\mid\!\sim \circ\Box F \to \circ H$	(2)(pax2)(mp)
(4)	$\mid\!\sim \Box F \to \circ H$	(3)(pax3)

Q.E.D.

We list some derived theorems of Σ_{TLA^*}, which will be used later. Derivations of these theorems are given in section A.1 in the appendix.

(T1)	$\vdash \Box F \leftrightarrow \Box\Box F$	(T2)	$\vdash \Box[A]_v \leftrightarrow \Box\Box[A]_v$
(T3)	$\vdash \Box[[A]_v]_v \leftrightarrow \Box[A]_v$	(T4)	$\vdash \Box[A]_v \rightarrow \Box[[A]_v]_w$
(T5)	$\vdash \Box[[A]_w]_v \rightarrow \Box[[A]_v]_w$	(T6)	$\vdash \Box F \rightarrow \Box[\circ F]_v$
(T7)	$\mid\!\sim \Box F \leftrightarrow F \wedge \circ\Box F$	(T8)	$\mid\!\sim \circ(F \wedge G) \leftrightarrow \circ F \wedge \circ G$

By rule (pre), every provable formula is also provable as a pre-formula. An important result for Σ_{TLA^*} shows that, conversely, $\mid\!\sim F$ implies $\vdash F$ for any formula F. Its rather technical proof requires a careful analysis of the derivations in Σ_{TLA^*} and is given in section A.2 in the appendix.

Theorem 2.8 *For any set $\mathcal{F}$ of formulas and any formula F:*

$$\mathcal{F} \vdash F \quad \textit{iff} \quad \mathcal{F} \mid\!\sim F \quad \textit{iff} \quad \mathcal{F} \mid\!\sim \circ F$$

2.4 Completeness of Σ_{TLA^*}

We will now set out to prove the completeness of Σ_{TLA^*}. Let us first note that compactness fails for pTLA*, just as it does for standard propositional linear-time temporal logic [57]:

Example 2.9 *Let $\mathcal{F} = \{\Box[v_i \rightarrow v'_{i+1}]_{v_i}, \Box(v_i \rightarrow w) : i \geq 0\}$. It is easy to see that $\mathcal{F} \models v_0 \rightarrow \Box w$, but we can clearly not derive $\mathcal{F} \vdash v_0 \rightarrow \Box w$, because this would require the infinitary invariant $\Box \bigvee_{i\geq 0} v_i$.*

We can therefore only hope for completeness when $\mathcal{F}$ is a finite set, and by theorem 2.7 it is enough to show that $\models F$ implies $\vdash F$.

Our completeness proof follows the standard approach [57] of constructing a model for a finite and consistent set of formulas. To do so, we have to assemble information about pre-formulas as well as formulas. The critical step in the proof is to show that all the essential information is contained in the formulas used for the construction; in particular, it is not necessary to induct on (proper) pre-formulas. This result, which may seem somewhat surprising, is due to the fact that the assumptions in a derivation $\mathcal{F} \vdash F$ contain only formulas. For a set $\mathcal{G}$ of pre-formulas, we denote by $\mathcal{G}^F$ the set of all formulas contained in $\mathcal{G}$. We also use $\mathcal{G}$ to denote the conjunction of all pre-formulas in $\mathcal{G}$; it will always be clear from the context whether we refer to the set or the pre-formula.

A set $\mathcal{G}$ of pre-formulas is called *inconsistent* if $\mid\!\sim \neg\mathcal{G}$, otherwise it is called *consistent*. Note that if $\mathcal{G}$ is consistent and A is any pre-formula, one of the sets $\mathcal{G} \cup \{A\}$ or $\mathcal{G} \cup \{\neg A\}$ is again consistent.

We inductively define a set $\tau(A)$ for any pre-formula A, as follows:

$$
\begin{array}{rclrcl}
\tau(v) & = & \{v\} & \tau(\neg A) & = & \{\neg A\} \cup \tau(A) \\
\tau(A \to B) & = & \{A \to B\} \cup \tau(A) \cup \tau(B) & \tau(\Box F) & = & \{\Box F\} \cup \tau(F) \\
\tau(\Box[A]_v) & = & \{\Box[A]_v, v, \circ v\} \cup \tau(A) & \tau(\circ F) & = & \{\circ F\}
\end{array}
$$

For a set $\mathcal{G}$, we define $\tau(\mathcal{G})$ as the union of all $\tau(A)$, for all pre-formulas A contained in $\mathcal{G}$. These definitions ensure that $\tau(\mathcal{G})$ is finite whenever $\mathcal{G}$ is finite.

We say that $\mathcal{G}$ is *complete* if it contains either A or $\neg A$, for every pre-formula A from $\tau(\mathcal{G})$. Because $\tau(\mathcal{G})$ is finite whenever $\mathcal{G}$ is, there exist only finitely many finite, consistent, and complete $\mathcal{G}^* \supseteq \mathcal{G}$; we call any such $\mathcal{G}^*$ a *completion* of $\mathcal{G}$. We note the following elementary facts about complete sets. The proofs of assertions 1 and 3 are standard, whereas the second assertion follows from the first and theorem 2.8.1 by propositional reasoning.

Proposition 2.10

1. *Assume that $\mathcal{G}$ is finite and consistent, and that $\mathcal{G}_1^*, \ldots, \mathcal{G}_n^*$ are all the different completions of $\mathcal{G}$. Then $\mid\!\sim \mathcal{G} \to \mathcal{G}_1^* \vee \ldots \vee \mathcal{G}_n^*$.*
2. *Assume that $\mathcal{F}$ is a finite and consistent set of formulas, and that $\mathcal{G}_1, \ldots, \mathcal{G}_n$ are all the different completions of $\mathcal{F}$. Then $\vdash \mathcal{F} \to \mathcal{G}_1^F \vee \ldots \vee \mathcal{G}_n^F$.*
3. *Assume that $\mathcal{G}$ is consistent and complete and that A, B are (pre-)formulas.*
   (a) *If $A \in \mathcal{G}$, $B \in \tau(\mathcal{G})$ and $\vdash A \to B$ or $\mid\!\sim A \to B$ then $B \in \mathcal{G}$.*
   (b) *If $A \to B \in \tau(\mathcal{G})$ then $A \to B \in \mathcal{G}$ iff $A \notin \mathcal{G}$ or $B \in \mathcal{G}$.*

Next, given a set $\mathcal{G}$ of pre-formulas, we define a set $\sigma(\mathcal{G})$ of formulas that, intuitively, transfer information from one state of the model under construction to the next one.

$$
\begin{array}{ll}
\sigma_1(\mathcal{G}) = \{F : \circ F \in \mathcal{G}\} & \sigma_2(\mathcal{G}) = \{\neg F : \neg\circ F \in \mathcal{G}\} \\
\sigma_3(\mathcal{G}) = \{\Box F : \Box F \in \mathcal{G}\} & \sigma_4(\mathcal{G}) = \{\neg\Box F : \neg\Box F \in \mathcal{G}, F \in \mathcal{G}\} \\
\sigma_5(\mathcal{G}) = \{\Box[A]_v : \Box[A]_v \in \mathcal{G}\} & \\
\sigma_6(\mathcal{G}) = \{\neg\Box[A]_v : \neg\Box[A]_v \in \mathcal{G} \text{ and} & \\
\qquad\qquad A \in \mathcal{G} \text{ or } \{v, \circ v\} \subseteq \mathcal{G} \text{ or } \{\neg v, \neg\circ v\} \subseteq \mathcal{G}\} & \\
\sigma(\mathcal{G}) = \sigma_1(\mathcal{G}) \cup \sigma_2(\mathcal{G}) \cup \sigma_3(\mathcal{G}) \cup \sigma_4(\mathcal{G}) \cup \sigma_5(\mathcal{G}) \cup \sigma_6(\mathcal{G}) &
\end{array}
$$

Lemma 2.11 *Assume that $\mathcal{G}$ is finite.*

1. $\mid\!\sim \mathcal{G} \to \circ\sigma(\mathcal{G})$.

2. *If $\mathcal{G}$ is consistent, then so is $\sigma(\mathcal{G})$.*

Proof.

1. By (T8), it is enough to show $\mid\!\sim \mathcal{G} \to \circ F$, for every formula $F \in \sigma(G)$. We distinguish the different cases in the definition of $\sigma(\mathcal{G})$.

   - For $F \in \sigma_1(\mathcal{G})$, we have $\circ F \in \mathcal{G}$, so the assertion follows by (pax0).
   - If $F \equiv \neg G \in \sigma_2(\mathcal{G})$, then $\neg\circ F \in \mathcal{G}$, and the assertion follows using (pax1).
   - If $F \equiv \Box G \in \sigma_3(\mathcal{G})$, we have $\Box G \in \mathcal{G}$; use (pax3) to prove the assertion.
   - If $F \equiv \neg\Box G \in \sigma_4(\mathcal{G})$, the definition ensures $\mid\!\sim \mathcal{G} \to G \wedge \neg\Box G$, and the assertion follows by (T7), (pax1), and propositional logic.
   - For $F \equiv \Box[A]_v \in \sigma_5(\mathcal{G})$, the assertion holds by (pax4).
   - If $F \equiv \neg\Box[A]_v \in \sigma_6(\mathcal{G})$, the definition and (pax0) yield $\mathcal{G} \to [A]_v \wedge \neg\Box[A]_v$, and the assertion follows by (pax4) and (pax1).

2. If $\sigma(\mathcal{G})$ is inconsistent, we have $\mid\!\sim \neg\sigma(\mathcal{G})$. We infer $\mid\!\sim \circ\neg\sigma(\mathcal{G})$ with the help of rule (nex). Using axiom (pax1) and propositional logic, the first assertion implies $\mid\!\sim \neg\mathcal{G}$, that is, $\mathcal{G}$ is inconsistent. Q.E.D.

Given a finite and consistent set $\mathcal{F}$ of formulas, we inductively define a forest $\mathcal{T}(\mathcal{F})$ of sets of pre-formulas as follows:

- The roots of $\mathcal{T}(\mathcal{F})$ are all different completions of $\mathcal{F}$.

- If $\mathcal{G}$ is a node in $\mathcal{T}(\mathcal{F})$ then its sons are all different completions of $\sigma(\mathcal{G})$.

It follows that every node $\mathcal{G}$ in $\mathcal{T}(\mathcal{F})$ is finite, consistent, and complete. Also, the sub-forest of $\mathcal{T}(\mathcal{F})$ whose roots are the sons of $\mathcal{G}$ is exactly $\mathcal{T}(\sigma\mathcal{G})$.

Lemma 2.12 *Assume that $\mathcal{F}$ is a finite and consistent set of formulas.*

1. *$\mathcal{T}(\mathcal{F})$ contains only finitely many different nodes $\mathcal{G}_1, \ldots, \mathcal{G}_n$.*

2. $\mid\!\sim \mathcal{G}_i^F \to \mathcal{G}_1 \vee \ldots \vee \mathcal{G}_n$, *for $i = 1, \ldots, n$.*

3. $\mid\!\sim \mathcal{G}_1^F \vee \ldots \vee \mathcal{G}_n^F \to \circ(\mathcal{G}_1^F \vee \ldots \vee \mathcal{G}_n^F)$.

4. $\vdash \mathcal{F} \to \Box(\mathcal{G}_1^F \vee \ldots \vee \mathcal{G}_n^F)$.

Proof.

1. The completions of a finite set $\mathcal{G}$ only contain pre-formulas from the set $\tau(\mathcal{G})$, which is also finite. On the other hand, the only pre-formulas in $\sigma(\mathcal{G})$ that are possibly not in $\tau(\mathcal{G})$ are of the form F or $\neg F$ such that $\mathcal{G}$ contains $\circ F$ or $\neg\circ F$, hence the number of $\circ$ operators decreases, which is possible only finitely often. Therefore, only finitely many different pre-formulas occur in $\mathcal{T}(\mathcal{F})$, and $\mathcal{T}(\mathcal{F})$ can contain only finitely many different nodes.

2. Consider the set $\mathcal{F}'$ of formulas from which node $\mathcal{G}_i$ was constructed—either the initial set $\mathcal{F}$ or the set $\sigma(\mathcal{G}_j)$ where $\mathcal{G}_j$ is the father of $\mathcal{G}_i$ in $\mathcal{T}(\mathcal{F})$. Proposition 2.10.1 implies $\mid\!\sim \mathcal{F}' \to \mathcal{G}_1 \vee \ldots \vee \mathcal{G}_n$ because all consistent completions of $\mathcal{F}'$ are contained in $\mathcal{T}(\mathcal{F})$. Moreover, we have $\mathcal{F}' \subseteq \mathcal{G}_i^F$ because $\mathcal{G}_i$ is a completion of $\mathcal{F}'$, and this implies $\mid\!\sim \mathcal{G}_i^F \to \mathcal{F}'$ by (pax0).

3. By lemma 2.11 we know $\mid\!\sim \mathcal{G}_j \to \circ\sigma(\mathcal{G}_j)$, for every node $\mathcal{G}_j$ of $\mathcal{T}(\mathcal{F})$. Proposition 2.10.2 ensures $\vdash \sigma(\mathcal{G}_j) \to \mathcal{G}_1^F \vee \ldots \vee \mathcal{G}_n^F$. Using (nex) and (pax2), we obtain $\mid\!\sim \mathcal{G}_j \to \circ(\mathcal{G}_1^F \vee \ldots \vee \mathcal{G}_n^F)$. Assertion 2 implies $\mid\!\sim \mathcal{G}_1^F \vee \ldots \vee \mathcal{G}_k^F \to \mathcal{G}_1 \vee \ldots \vee \mathcal{G}_n$, and the assertion follows by propositional reasoning.

4. Let I denote the formula $\mathcal{G}_1^F \vee \ldots \vee \mathcal{G}_n^F$. From assertion 3 and rule (sq) we conclude $\vdash \Box[I \to \circ I]_I$, hence $\vdash I \to \Box I$ by axiom (ax3). On the other hand, proposition 2.10.2 implies $\vdash \mathcal{F} \to I$, which proves the assertion. Q.E.D.

We will construct a model for $\mathcal{F}$ from suitable paths in $\mathcal{T}(\mathcal{P})$. Let us call a path $\mathcal{G}_0, \mathcal{G}_1, \ldots$ *complete* iff it satisfies the two following conditions, for all $i \in \mathbb{N}$:

- If $\neg\Box F \in \mathcal{G}_i$ then $\neg F \in \mathcal{G}_j$ for some $j \geq i$.
- If $\neg\Box[A]_v \in \mathcal{G}_i$ then for some $j \geq i$, $\neg A \in \mathcal{G}_j$ and either $\{v, \neg\circ v\} \subseteq \mathcal{G}_j$ or $\{\neg v, \circ v\} \subseteq \mathcal{G}_j$.

Lemma 2.13 *Assume that $\mathcal{F}$ is a finite and consistent set of formulas. Then $\mathcal{T}(\mathcal{F})$ contains a complete path.*

Proof. Let again $\mathcal{G}_1, \ldots, \mathcal{G}_n$ denote the nodes of $\mathcal{T}(\mathcal{F})$, and write I for the formula $\mathcal{G}_1^F \vee \ldots \vee \mathcal{G}_n^F$.

We first prove that for any formula $\neg\Box F \in \mathcal{F}$ there is some node $\mathcal{G}$ in $\mathcal{T}(\mathcal{F})$ that contains $\neg F$. Suppose not. Then, in particular, every root $\mathcal{W}$ of $\mathcal{T}(\mathcal{F})$ contains $\neg\Box F$ and F (because $F \in \tau(\mathcal{F})$ and $\mathcal{W}$ is a completion of $\mathcal{F}$), hence $\neg\Box F \in \sigma(\mathcal{W})$. Inductively, it follows that $\{\neg\Box F, F\} \subseteq \mathcal{G}_i^F$ holds for all $i = 1, \ldots, n$. So, (ax0) gives $I \to F$, hence also $I \vdash \Box F$, using rule (alw). By theorem 2.7, we conclude $\vdash \Box I \to \Box F$. Lemma 2.12.4 yields $\vdash \mathcal{F} \to \Box F$, but on the other hand we have $\vdash \mathcal{F} \to \neg\Box F$ because $\neg\Box F \in \mathcal{F}$, contradicting the assumption that $\mathcal{F}$ is consistent.

Similarly, we show that for any formula $\neg\Box[A]_v \in \mathcal{F}$ there is some node $\mathcal{G}$ in $\mathcal{T}(\mathcal{F})$ that contains $\neg A$ and either $\{v, \neg\circ v\}$ or $\{\neg v, \circ v\}$. Suppose not. Then an argument analogous to the one above establishes that every node $\mathcal{G}_j$ contains A or $\{v, \circ v\}$ or $\{\neg v, \neg\circ v\}$. By axiom (pax0), this shows $\mid\!\sim \mathcal{G}_j \rightarrow [A]_v$, for $j = 1, \ldots, n$. Lemma 2.12.2 implies $\mid\!\sim I \rightarrow [A]_v$, and by (ax1) and (pre), a fortiori $\mid\!\sim \Box I \rightarrow [A]_v$. Using rule (sq) and (ax4), this shows $\vdash \Box[\Box I]_v \rightarrow \Box[[A]_v]_v$, and (T3) implies that $\vdash \Box[\Box I]_v \rightarrow \Box[A]_v$. But as above we have $\vdash \mathcal{F} \rightarrow \Box I$, and thus also $\vdash \mathcal{F} \rightarrow \Box[\Box I]_v$ by (ax2), which proves $\vdash \mathcal{F} \rightarrow \Box[A]_v$. On the other hand, we know $\vdash \mathcal{F} \rightarrow \neg\Box[A]_v$ by assumption and reach a contradiction.

These two claims ensure that for every node $\mathcal{G}$ in $\mathcal{T}(\mathcal{F})$ that contains either $\neg\Box F$ or $\neg\Box[A]_v$ there exists some node $\mathcal{G}'$ below $\mathcal{G}$ that satisfies the condition from the definition of a complete path. For if $\mathcal{G}$ itself does not satisfy the condition, the formula is contained in $\sigma(\mathcal{G})$, hence $\mathcal{T}(\sigma(\mathcal{G}))$, which is just the subforest of $\mathcal{T}(\mathcal{F})$ whose roots are the sons of $\mathcal{G}$, contains a node as required.

The assertion is now proved by fixing some order on the finite set of formulas $\neg\Box F$ and $\neg\Box[A]_v$ that occur in $\mathcal{T}(\mathcal{F})$ and an iterative construction that constructs a complete path piecewise by repeatedly considering the eventuality formulas in the chosen order. The details of this construction are standard [39, 57]. Q.E.D.

Lemma 2.14 *Assume that $\mathcal{F}$ is a finite and consistent set of formulas and that $\mathcal{G}_0, \mathcal{G}_1, \ldots$ is a complete path in $\mathcal{T}(\mathcal{F})$. For every $i \in \mathbb{N}$, the following assertions hold:*

1. *If $\circ F \in \tau(\mathcal{G}_i)$ then $\circ F \in \mathcal{G}_i$ iff $F \in \mathcal{G}_{i+1}$.*
2. *If $\Box F \in \tau(\mathcal{G}_i)$ then $\Box F \in \mathcal{G}_i$ iff $F \in \mathcal{G}_j$ for all $j \geq i$.*
3. *If $\Box[A]_v \in \tau(\mathcal{G}_i)$ then $\Box[A]_v \in \mathcal{G}_i$ iff for all $j \geq i$, $A \in \mathcal{G}_j$ or $\{v, \circ v\} \subseteq \mathcal{G}_j$ or $\{\neg v, \neg\circ v\} \subseteq \mathcal{G}_j$.*

Proof.

1. If $\circ F \in \mathcal{G}_i$ then $F \in \sigma(\mathcal{G}_i)$ and therefore $F \in \mathcal{G}_{i+1}$, which is a completion of $\sigma(\mathcal{G}_i)$.

   If $\circ F \notin \mathcal{G}_i$ then $\neg\circ F \in \mathcal{G}_i$ (because $\mathcal{G}_i$ is a complete set), so $\neg F \in \sigma(\mathcal{G}_i)$, and again $\neg F \in \mathcal{G}_{i+1}$. But the consistency of $\mathcal{G}_{i+1}$ implies $F \notin \mathcal{G}_{i+1}$.

2. Assume $\Box F \in \mathcal{G}_i$. Then we have $F \in \tau(\mathcal{G}_i)$, and because of $\vdash \Box F \rightarrow F$ (ax1) and proposition 2.10.3, it follows that $F \in \mathcal{G}_i$. Moreover, $\Box F \in \sigma(\mathcal{G}_i)$ and therefore $\Box F \in \mathcal{G}_{i+1}$. Inductively, we conclude that $F \in \mathcal{G}_j$ holds for all $j \geq i$.

   Conversely, if $F \in \mathcal{G}_j$ for all $j \geq i$ then the definition of a complete path and the consistency of the $\mathcal{G}_j$ ensure that $\neg\Box F \in \mathcal{G}_i$ cannot hold. The assumption $\Box F \in \tau(\mathcal{G}_i)$ and the fact that $\mathcal{G}_i$ is a complete pair imply $\Box F \in \mathcal{G}_i$.

3. Assume $\Box[A]_v \in \mathcal{G}_i$. Then $\{A, v, \circ v\} \subseteq \tau(\mathcal{G}_i)$, and by $\mid\!\sim \Box[A]_v \rightarrow [A]_v$ (pax4) and proposition 2.10.3, the assertion follows for $j = i$ by the completeness and consistency of $\mathcal{G}_i$ and propositional logic. Moreover, $\Box[A]_v \in \sigma(\mathcal{G}_i)$ and therefore $\Box[A]_v \in \mathcal{G}_{i+1}$. Inductively, the assertion follows for all $j \geq i$.

   Conversely, if $A \in \mathcal{G}_j$ or $\{v, \circ v\} \subseteq \mathcal{G}_j$ or $\{\neg v, \neg \circ v\} \subseteq \mathcal{G}_j$ holds for all $j \geq i$, the consistency of the $\mathcal{G}_j$ implies that there can be no $j \geq i$ such that $\neg A \in \mathcal{G}_j$ and either $\{v, \neg \circ v\} \subseteq \mathcal{G}_j$ or $\{\neg v, \circ v\} \subseteq \mathcal{G}_j$. Using the definition of a complete path, it follows that $\neg\Box[A]_v \in \mathcal{G}_i$ cannot hold, hence $\Box[A]_v \in \mathcal{G}_i$. Q.E.D.

We can now prove that, given a finite and consistent set $\mathcal{F}$ of formulas, a model of $\mathcal{F}$ can be constructed from $\mathcal{T}(\mathcal{F})$.

Lemma 2.15 *For every finite and consistent set $\mathcal{F}$ of formulas there is a behavior σ such that $\sigma \models F$ holds for all $F \in \mathcal{F}$.*

Proof. Assume that $\mathcal{F}$ is a finite and consistent set of formulas. Construct $\mathcal{T}(\mathcal{F})$ and choose some complete path $\mathcal{G}_0, \mathcal{G}_1, \dots$ in $\mathcal{T}(\mathcal{F})$; by lemma 2.13, such a path exists. Now define the behavior $\sigma = s_0 s_1 \dots$ by $s_i(v) = \mathrm{tt}$ iff $v \in \mathcal{G}_i$, for every $v \in \mathcal{V}$.

By induction on the structure of pre-formulas, we prove that for all pre-formulas A and all $i \in \mathbb{N}$, if $A \in \tau(\mathcal{G}_i)$ then $\sigma[i\,..] \mid\!\approx A$ iff $A \in \mathcal{G}_i$.

Because of $\mathcal{F} \subseteq \mathcal{G}_0$ and $F \in \tau(\mathcal{F}) = \tau(\mathcal{G}_0)$ for every $F \in \mathcal{F}$, this in particular implies $\sigma \models F$ for all formulas $F \in \mathcal{F}$.

The inductive proof of the assertion is again standard; we only give a few cases:

$\Box[A]_v$: Assume $\Box[A]_v \in \tau(\mathcal{G}_i)$. Therefore, either $\Box[A]_v \in \mathcal{G}_i$ or $\neg\Box[A]_v \in \mathcal{G}_i$. In the former case, lemma 2.14.3 implies that, for all $j \geq i$, $A \in \mathcal{G}_j$ or $\{v, \circ v\} \subseteq \mathcal{G}_j$ or $\{\neg v, \neg \circ v\} \in \mathcal{G}_j$. By induction hypothesis, this implies that $\sigma[j\,..] \mid\!\approx A$ or $s_j(v) = s_{j+1}(v)$ for all $j \geq i$, and therefore $\sigma[i\,..] \models \Box[A]_v$.

If $\neg\Box[A]_v \in \mathcal{G}_i$, the definition of a complete path ensures that $\neg A \in \mathcal{G}_j$ and either $\{v, \neg \circ v\} \subseteq \mathcal{G}_j$ or $\{\neg v, \circ v\} \subseteq \mathcal{G}_j$ holds for some $j \geq i$, and the induction hypothesis implies $\sigma[i\,..] \models \neg\Box[A]_v$.

$\circ F$: Assume $\circ F \in \tau(\mathcal{G}_i)$. By lemma 2.14.1, $\circ F \in \mathcal{G}_i$ iff $F \in \mathcal{G}_{i+1}$ iff (by induction hypothesis) $\sigma[i+1\,..] \models F$ iff $\sigma[i\,..] \mid\!\approx \circ F$. Q.E.D.

The completeness theorem now follows as a corollary to lemma 2.15.

Theorem 2.16 (Completeness) *For every formula F, if $\models F$ then $\vdash F$.*

Proof. Assume $\models F$. Then $\sigma \models \neg F$ holds for no behavior σ, and lemma 2.15 implies that $\{\neg F\}$ is inconsistent, that is $\mid\!\sim \neg\neg F$, from which $\vdash F$ follows by theorem 2.8 and propositional logic. Q.E.D.

(hx0)	F for F tautological	(hx7)	$\Box[A]_v$ for A tautological
(hx1)	$\Box F \to F$	(hx8)	$\Box[\circ\neg F \leftrightarrow \neg\circ F]_v$
(hx2)	$\Box F \to \Box[F]_v$	(hx9)	$\Box[\circ(F \to G) \to (\circ F \to \circ G)]_v$
(hx3)	$\Box F \to \Box[\circ\Box F]_v$	(hx10)	$\Box[\Box[A]_v \to [A]_v]_w$
(hx4)	$\Box[F \to \circ F]_F \to (F \to \Box F)$	(hx11)	$\Box[A]_v \to \Box[\circ\Box[A]_v]_w$
(hx5)	$\Box[A \to B]_v \to (\Box[A]_v \to \Box[B]_v)$	(hx12)	$\Box[[A]_v \wedge \circ\Box[A]_v \to \Box[A]_v]_w$
(hx6)	$\Box[v'\neg v)]_v$	(hx13)	$\Box[\circ\Box F \to \Box[\circ F]_v]_w$
(hmp)	$F, F \to G \overset{h}{\vdash} G$	(alw)	$F \overset{h}{\vdash} \Box F$

Figure 2.2: The proof system $\Sigma^h_{\mathrm{TLA}^*}$.

2.5 An Alternative Axiomatization

The only rule of Σ_{TLA^*} that allows the derivation $|\!\sim A$ of a proper preformula A to be used in the derivation of a formula F is rule (sq). This observation suggests the definition of a proof system that does not rely on an auxiliary relation $|\!\sim$ by "boxing" the axioms (pax0)–(pax5) and rephrasing rules (pre), (nex), and (pmp) accordingly. The proof system $\Sigma^h_{\mathrm{TLA}^*}$ shown in figure 2.2 is based on this idea and some further simplifications. The following theorems and rules can be derived in $\Sigma^h_{\mathrm{TLA}^*}$; derivations are shown in section A.3 in the appendix.

(H1)	$\Box[P]_v, \Box[P \to Q]_v \overset{h}{\vdash} \Box[Q]_v$	(H2)	$F \overset{h}{\vdash} \Box[F]_v$
(H3)	$\Box[P \to Q]_v, \Box[Q \to R]_v \overset{h}{\vdash} \Box[P \to R]_v$		
(H4)	$\Box[[P]_v \to P]_v$	(H5)	$\Box[\Box F \to \circ\Box F]_v$

We now show that the provability relations $\vdash$ and $\overset{h}{\vdash}$ agree (where $\mathcal{F} \overset{h}{\vdash} F$ is defined in the obvious way). In particular, $\Sigma^h_{\mathrm{TLA}^*}$ is also sound and complete. It is therefore a matter of taste and convenience which axiomatization to use. Although $\Sigma^h_{\mathrm{TLA}^*}$ is more attractive from a theoretical point of view because all derivations are carried out at the formula level, it is probably easier to find actual derivations in the heterogeneous system Σ_{TLA^*}.

Theorem 2.17 *For any set $\mathcal{F}$ of formulas and any formula F, $\mathcal{F} \vdash F$ iff $\mathcal{F} \overset{h}{\vdash} F$.*

Proof.

"only if": By induction on the length of the assumed derivation in Σ_{TLA^*}, we prove for all formulas F and pre-formulas A that $\mathcal{F} \overset{h}{\vdash} F$ whenever $\mathcal{F} \vdash F$ and that $\mathcal{F} \overset{h}{\vdash} \Box[A]_v$, for any atomic proposition v, whenever $\mathcal{F} |\!\sim A$.

If $F \in \mathcal{F}$ or if F it is an instance of (ax0), (ax1), (ax3), (ax4) or (ax5) then the assertion holds trivially because these axioms are also contained in $\Sigma^h_{\text{TLA}^*}$. Axiom (ax2) is derived in $\Sigma^h_{\text{TLA}^*}$ as follows:

(1)	$\Box[\Box F \rightarrow \circ\Box F]_F$	(H4)(prop)
(2)	$\Box F \rightarrow \Box\Box F$	(1)(hx4)(mp)
(3)	$\Box\Box F \rightarrow \Box[\Box F]_v$	(hx2)
(4)	$\Box F \rightarrow \Box[\Box F]_v$	(prop)(2)(3)

If the last step in the derivation of $\mathcal{F} \vdash F$ is an application of (mp) to previously derived formulas G and $G \rightarrow F$ then by induction hypothesis we have $\mathcal{F} \overset{h}{\vdash} G$ and $\mathcal{F} \overset{h}{\vdash} G \rightarrow F$, so $\mathcal{F} \overset{h}{\vdash} F$ follows by rule (hmp).

If the last step in the derivation of $\mathcal{F} \vdash F$ is an application of (sq) to some previously derived pre-formula A (so F is $\Box[A]_v$) then by the induction hypothesis for the second assertion we already have $\mathcal{F} \overset{h}{\vdash} \Box[A]_v$.

The second assertion is trivial if the last step in the derivation of $\mathcal{F} \mathrel{|\!\sim} A$ is an instance of (pax0), (pax1), (pax2) or (pax5) because $\Sigma^h_{\text{TLA}^*}$ contains corresponding axioms. The case of (pax3) is taken care of by (H4). As for (pax4), it could clearly be replaced by

(pax4a)	$\mathrel{|\!\sim} \Box[A]_v \rightarrow [A]_v$
(pax4b)	$\mathrel{|\!\sim} \Box[A]_v \rightarrow \circ\Box[A]_v$
(pax4c)	$\mathrel{|\!\sim} [A]_v \wedge \circ\Box[A]_v \rightarrow \Box[A]_v$

without changing the set of pre-formulas derivable in Σ_{TLA^*}. The axioms (hx10) and (hx12) directly correspond to (pax4a) and (pax4c), so it remains to consider the case of (pax4b), by the following derivation:

(1)	$\Box[A]_v \rightarrow \Box[\circ\Box[A]_v]_w$	(hx11)
(2)	$\Box[\Box[A]_v \rightarrow \Box[\circ\Box[A]_v]_w]_w$	(H2)(1)
(3)	$\Box[\Box[\circ\Box[A]_v]_w \rightarrow [\circ\Box[A]_v]_w]_w$	(hx10)
(4)	$\Box[\Box[A]_v \rightarrow [\circ\Box[A]_v]_w]_w$	(H3)(2)(3)
(5)	$\Box[[\circ\Box[A]_v]_w \rightarrow \circ\Box[A]_v]_w$	(H4)
(6)	$\Box[\Box[A]_v \rightarrow \circ\Box[A]_v]_w$	(H3)(4)(5)

Considering the rules, the case of (pmp) is handled by the induction hypothesis and (H1). If the last step in the derivation of $\mathcal{F} \mathrel{|\!\sim} A$ is an application of (pre), then A is actually a formula and has already been derived, so we may assume $\mathcal{F} \overset{h}{\vdash} A$ by induction hypothesis. We obtain $\mathcal{F} \overset{h}{\vdash} \Box[A]_v$ by (H2).

If the last step is an application of (nex), then A is $\circ F$, for some previously derived formula F, and by induction hypothesis we may assume $\mathcal{F} \overset{h}{\vdash} F$. We continue as follows:

(1)	F	(ind.hyp.)
(2)	$\Box F$	(alw)(1)
(3)	$\Box[\circ\Box F]_v$	(2)(hx3)(hmp)
(4)	$\Box[\circ\Box F \to \Box[\circ F]_v]_v$	(hx13)
(5)	$\Box[\Box[\circ F]_v]_v$	(H1)(3)(4)
(6)	$\Box[\Box[\circ F]_v \to [\circ F]_v]_v$	(hx10)
(7)	$\Box[[\circ F]_v]_v$	(H1)(5)(6)
(8)	$\Box[\circ F]_v$	(7)(H4)(H1)

"if": The proof is again by induction on the assumed derivation of $\mathcal{F} \overset{h}{\vdash} F$. The cases of (hx0), (hx1), (hx4), (hx5), and (hx6) are trivial because Σ_{TLA^*} contains the same axioms. For (hx7), (hx8), (hx9), (hx10), (hx12), and (hx13), the proof uses the corresponding axioms of Σ_{TLA^*} and rule (sq). For (hmp) and (alw), the assertion follows from the induction hypothesis and rules (mp) and (alw), which is a derived rule in Σ_{TLA^*}.

The axiom (hx2) is derived in Σ_{TLA^*} as follows:

(1)	$\mid\!\sim \Box F \to F$	(ax1)(pre)
(2)	$\vdash \Box[\Box F \to F]_v$	(sq)(1)
(3)	$\vdash \Box[\Box F]_v \to \Box[F]_v$	(2)(ax4)(mp)
(4)	$\vdash \Box F \to \Box[\Box F]_v$	(ax2)
(5)	$\vdash \Box F \to \Box[F]_v$	(prop)(3)(4)

The derivation of (hx3) is similar, using (pax3) instead of (ax1). The derivation of (hx11) is very similar to that of (T4) and is omitted. Q.E.D.

Chapter 3

Stuttering ω-regular languages

3.1 Overview

Automata that operate on infinite objects such as ω-words, trees or graphs [98, 99] have been important tools for the analysis of temporal and related logics since the pioneering work by Büchi, Muller, McNaughton, Rabin, Vardi, and Wolper [25, 72, 79, 87, 103]. The correspondence between, say, linear-time temporal logic and automata over ω-words is established by identifying states with subsets of $\mathcal{V}$ (that contain those propositions that are true of the state). Behaviors are then regarded as ω-words over the alphabet $2^{\mathcal{V}}$. In this way, automata can serve as acceptors for temporal structures, and this view has been crucial in proving completeness and decidability theorems. We will use automata-theoretic constructions in chapter 4 to prove a completeness result for an extension of pTLA* by quantification over flexible propositions. However, in view of the invariance of pTLA* formulas with respect to stuttering equivalence (cf. theorem 2.5), we will restrict attention to stuttering-invariant ω-regular languages.

The main contribution of this chapter is a characterization of the class of ω-regular languages that are invariant under stuttering equivalence, by structural conditions on Büchi automata. To this end, we find it convenient to introduce a variation of the standard definition of Büchi automata whose acceptance condition is defined in terms of the transitions taken during a run rather than in terms of the locations that have been visited. We reestablish some classical automata constructions that will be needed in chapter 4, and in particular a variant of Safra's determinization procedure for these "edge automata". We will then show how to construct, for any given Büchi edge automaton $\mathcal{A}$, another automaton $\mathcal{A}_V^{sc}$ that defines the V-stuttering closure of the language defined by $\mathcal{A}$. This construction will be instrumental in the proof of our characterization theorem of stuttering invariant languages.

We use a presentation of ω-automata that is particularly suitable for establishing the desired correspondence with pTLA*. In particular, we leave the alphabet implicit and define the transition conditions via propositional formulas. Of course, this presentation could easily be recast in more conventional terms: the alphabet associated with an automaton $\mathcal{A}$ is just 2^V where V is the set of atomic propositions that occur in $\mathcal{A}$'s transition conditions.

3.2 Edge Automata

We assume a fixed denumerable set $\mathcal{V}$ of atomic propositions and let $\mathcal{L}(\mathcal{V})$ denote the set of Boolean formulas over $\mathcal{V}$.

Definition 3.1 (edge automata) *An* edge automaton $\mathcal{A} = (Q, q_0, \delta, C)$ *is given by*

- *a finite set* Q *of* locations[1],
- *an* initial location $q_0 \in Q$,
- *a* transition function $\delta : Q \times Q \to \mathcal{L}(\mathcal{V})$ *that associates a Boolean formula* $\delta(q, r)$ *over* $\mathcal{V}$ *with any pair* $q, r \in Q$ *of locations,*
- *and an* acceptance condition C *of one of the following forms:*

  Büchi. $C : Q \times Q \to \mathcal{L}(\mathcal{V})$ *associates a Boolean formula* $C(q, r)$ *with every transition (pair of states).*

  Rabin, Streett. $C = \{(R_1, G_1), \ldots, (R_m, G_m)\}$ *is a set of pairs* (R_i, G_i) *where* $R_i, G_i : Q \times Q \to \mathcal{L}(\mathcal{V})$ *again associate Boolean formulas with transitions.*

The automaton $\mathcal{A}$ *is* deterministic *if for all locations* $q \in Q$ *the assertions labelling the transitions from* q *are exhaustive and mutually exclusive, that is, if*

$$\models \bigvee_{r \in Q} \delta(q, r) \qquad \textit{and} \qquad \models \neg(\delta(q, r) \wedge \delta(q, s)) \quad \textit{for } r, s \in Q, r \neq s$$

It is loop-free *if*

$$\models \neg\delta(q, q)$$

holds for all $q \in Q$.

[1] We avoid the conventional term *state* in order to distinguish automata locations from the states of a temporal interpretation.

A run *of $\mathcal{A}$ over a behavior* $\sigma = s_0 s_1 \ldots$ *is an infinite sequence* $\rho = r_0 r_1 \ldots$ *of locations* $r_i \in Q$ *such that* $r_0 = q_0$ *and* $s_i \models \delta(r_i, r_{i+1})$ *holds for all* $i \in \mathbb{N}$. *The run* ρ *is* accepting *if it satisfies the automaton's acceptance condition, depending on the type of* $\mathcal{A}$*:*

Büchi. ρ *contains infinitely many accepting transitions, i.e. there are infinitely many* $i \in \mathbb{N}$ *such that* $s_i \models C(r_i, r_{i+1})$.

Rabin. *For some* $j \in \{1, \ldots, m\}$, *the "red" transitions in* R_j *occur only finitely often, but some "green" transition in* G_j *occurs infinitely often: there are only finitely many* $i \in \mathbb{N}$ *such that* $s_i \models R_j(r_i, r_{i+1})$, *but there are infinitely many* $i \in \mathbb{N}$ *such that* $s_i \models G(r_i, r_{i+1})$.

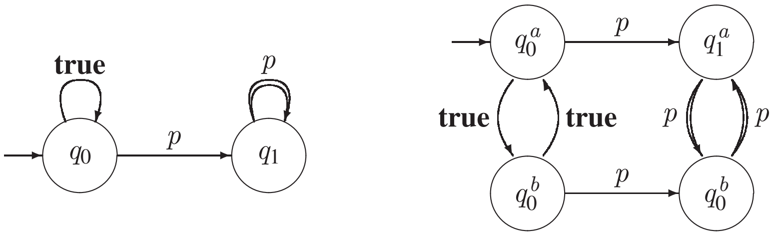
Streett. *For every* $j \in \{1, \ldots, m\}$, *if there are infinitely many "red" transitions in* R_j *then infinitely many "green" transitions in* G_j *must also occur: either there are only finitely many* $i \in \mathbb{N}$ *such that* $s_i \models R_j(r_i, r_{i+1})$ *or there are infinitely many* $i \in \mathbb{N}$ *such that* $s_i \models G(r_i, r_{i+1})$.

A behavior σ *is accepted by* $\mathcal{A}$ *if there exists some accepting run of* $\mathcal{A}$ *over* σ. *The* language $\mathcal{L}(\mathcal{A})$ *of* $\mathcal{A}$ *consists of those behaviors that are accepted by* $\mathcal{A}$. *Two edge automata* $\mathcal{A}$ *and* $\mathcal{B}$ *are* equivalent *iff* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.

Definition 3.1 is a variant of the standard definition of ω-automata over behaviors where acceptance conditions are defined in terms of transitions instead of the more conventional definition in terms of locations. It would perhaps be more natural to define the acceptance conditions merely via sets of edges, say $C \subseteq Q \times Q$ for a Büchi automaton. This is a special case of our definition when only **true** and **false** are allowed as formulas in the acceptance conditions. We have chosen a more liberal definition for technical reasons (mainly to simplify the determinization construction of section 3.3), although it can be shown that the restriction to the special case would not change the class of definable languages.

It is not hard to encode an edge automaton as a traditional ω-automaton whose locations are (roughly) pairs of locations of the edge automaton; several copies of the target location have to be created when the edge automaton may have taken an accepting transition. Conversely, a Büchi ω-automaton can be represented as a Büchi edge automaton whose accepting transitions are those that, say, leave an accepting location of the original automaton, and similarly for the other automaton types. Therefore, edge automata define the same class of (ω-regular) languages as ω-automata, although edge automata may be more succinct.

For automata locations q, r and state s, we sometimes write $q \xrightarrow{s} r$ instead of $s \models \delta(q, r)$ to indicate that a transition from q to r is possible in state s. Similarly, we write $q \stackrel{\tau}{\Rightarrow} r$, for a finite sequence $\tau = t_1 \ldots t_n$ of states if the input τ can take



(a) A Büchi automaton. (b) Loop-free variant.

Figure 3.1: Pictorial representation of Büchi edge automata.

the automaton from q to r, that is, if there exist locations $q_1, \ldots, q_{n+1}$ such that $q_1 = q$, $q_{n+1} = r$, and $q_i \xrightarrow{t_i} q_{i+1}$ for all $i \in \{1, \ldots, n\}$ (we sometimes say that there is a τ-path from q to r). If $E : Q \times Q \to \mathcal{L}(\mathcal{V})$ is an edge labelling, we write $q \stackrel{\tau}{\Rightarrow}_E r$ if there exists a τ-path from q to r such that $t_i \models E(q_i, q_{i+1})$ holds for at least one transition in the path.

We will mostly consider Büchi edge automata, which we represent pictorially as in figure 3.1. We usually indicate the acceptance condition by edges drawn with double lines, meaning that $C(q,r) \equiv \delta(q,r)$ for these transitions, and $C(q,r) \equiv$ **false** otherwise. For example, the automaton of figure 3.1(a) accepts precisely those behaviors that satisfy the temporal formula $\Diamond\Box p$.

Definition 3.1 introduces a class of *loop-free* automata where no transition leads back to its source location. It is easy to see that for every edge automaton with n locations there is an equivalent loop-free edge automaton of the same type with at most $2n$ locations, obtained by copying locations that carry a loop. For example, figure 3.1(b) shows a loop-free variant of the Büchi automaton shown in figure 3.1(a).

The standard results for ω-automata apply essentially unchanged to edge automata. For example, deterministic Büchi edge automata are strictly less expressive than nondeterministic ones. In particular, there is no deterministic Büchi automaton equivalent to the automaton of figure 3.1: Thomas [98] gives a proof of the corresponding result for standard Büchi automata, which equally applies to edge automata. We will now show that nondeterministic Büchi edge automata are as expressive as Streett automata.

Theorem 3.2 *For every Streett automaton $\mathcal{A}$ one can construct an equivalent Büchi automaton $\mathcal{B}$.*

Proof. The idea of the construction, which is attributed to Vardi in [88], is to exploit the temporal logic equivalence

$$(\Box\Diamond R \to \Box\Diamond G) \;\leftrightarrow\; \Diamond\Box(R \to \Diamond G)$$

and the conjunctivity of the formula on the right-hand side. The Büchi automaton $\mathcal{B}$ at some point nondeterministically decides to start recording "red" and "green" transitions of the Streett automaton. It makes an accepting transition whenever every "red" transition has been matched by a corresponding "green" transition, and then restarts its bookkeeping.

Formally, let $\mathcal{A} = (Q^{\mathcal{A}}, q_0^{\mathcal{A}}, \delta^{\mathcal{A}}, C^{\mathcal{A}})$ be the given Streett automaton with acceptance condition $C^{\mathcal{A}} = \{(R_1, G_1), \ldots, (R_m, G_m)\}$. The nondeterministic Büchi automaton $\mathcal{B} = (Q^{\mathcal{B}}, q_0^{\mathcal{B}}, \delta^{\mathcal{B}}, C^{\mathcal{B}})$ is defined as follows:

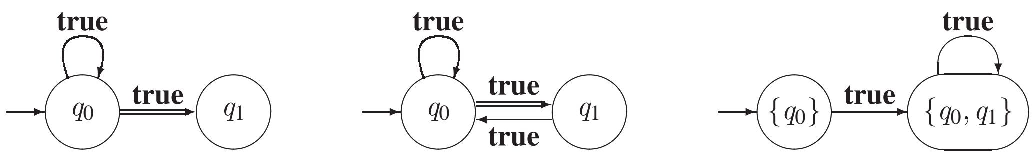
- $Q^{\mathcal{B}} \triangleq Q^{\mathcal{A}} \times (2^{\{1,\ldots,m\}} \cup \{\bot\}) \times (2^{\{1,\ldots,m\}} \cup \{\bot\})$.
- $q_0^{\mathcal{B}} \triangleq (q_0^{\mathcal{A}}, \bot, \bot)$.
- $\delta^{\mathcal{B}}((q,R,G),(q',R',G'))$ is defined by cases, as follows:
  1. $\delta^{\mathcal{B}}((q,\bot,\bot),(q',\bot,\bot)) \equiv \delta^{\mathcal{A}}(q,q')$
  2. $\delta^{\mathcal{B}}((q,\bot,\bot),(q',\emptyset,\emptyset)) \equiv \delta^{\mathcal{A}}(q,q')$
  3. $\delta^{\mathcal{B}}((q,R,G),(q',R',G')) \equiv \delta^{\mathcal{A}}(q,q') \wedge \mathcal{R}(q,q',R,R') \wedge \mathcal{G}(q,q',G,G')$ for $R,G,R',G' \neq \bot$, $R \subseteq R'$, $G \subseteq G'$, and $R' \not\subseteq G'$
  4. $\delta^{\mathcal{B}}((q,R,G),(q',\emptyset,\emptyset)) \equiv \delta^{\mathcal{A}}(q,q') \wedge \bigvee_{\substack{R',G' \subseteq \{1,\ldots,m\} \\ R \subseteq R', G \subseteq G', R' \subseteq G'}} \mathcal{R}(q,q',R,R') \wedge \mathcal{G}(q,q',G,G')$
  5. $\delta^{\mathcal{B}}((q,R,G),(q',R',G')) \equiv \textbf{false}$ otherwise

  where $\mathcal{R}(q,q',R,R')$ is defined by (for $R \subseteq R' \subseteq \{1,\ldots,m\}$)

  $$\mathcal{R}(q,q',R,R') \equiv \bigwedge_{j \in R' \setminus R} R_j(q,q') \wedge \bigwedge_{j \notin R'} \neg R_j(q,q')$$

  and similarly for $\mathcal{G}(q,q',G,G')$.
- $C^{\mathcal{B}}((q,R,G),(q',R',G')) \equiv \begin{cases} \delta^{\mathcal{B}}((q,R,G),(q',R',G')) & \text{if } R' = G' = \emptyset \\ \textbf{false} & \text{otherwise} \end{cases}$

Informally, transitions of type (1) occur during an initial prefix of the run where "red" and "green" transitions of $\mathcal{A}$ are ignored. Nondeterministically, $\mathcal{B}$ decides to start recording occurrences of such transitions by following a transition of type (2). After that, only transitions of type (3) and (4) can occur. Transitions of type (3) add indexes for which "red" and "green" $\mathcal{A}$-transitions have occurred to the R- and G-components of the locations of $\mathcal{B}$. When every occurrence of a transition in R_j has been matched by some transition in G_j, the second and third components of the locations of $\mathcal{B}$ are reset, via a transition of type (4), to $\emptyset$. Observe that the only additional non-determinism introduced by this construction is the choice between transitions of type (1) and (2). In particular, if $\mathcal{A}$ is deterministic then $\mathcal{B}$ is "deterministic in the limit", that is, all transitions following an accepting transition are deterministic.



(a) Automaton $\mathcal{A}$ (b) Automaton $\mathcal{B}$ (c) Subset automaton

Figure 3.2: Applying the subset construction to Büchi edge automata.

Suppose that some behavior $\sigma = s_0 s_1 \dots$ is accepted by $\mathcal{A}$, via run $\rho = r_0 r_1 \dots$. By the Streett acceptance condition and the equivalence shown above, there must be some $k \geq 0$ such that for all $p \geq k$ and all $j \in \{1, \dots, m\}$, whenever $s_p \models R_j(r_p, r_{p+1})$ there is some $q \geq p$ such that $s_q \models G_j(r_q, r_{q+1})$. Simulate ρ by a run of $\mathcal{B}$ that initializes R and G to $\emptyset$ at time k (via a transition of type 2) and then performs transitions of types (3) and (4). The above observation implies that R and G are infinitely often reset to $\emptyset$ via a transition of type (4), therefore $\mathcal{B}$ accepts σ.

Conversely, suppose that behavior σ is accepted by $\mathcal{B}$ via run $\tau = t_0 t_1 \dots$, and let $\rho = r_0 r_1 \dots$ be the corresponding run of $\mathcal{A}$ (that is, where every location of $\mathcal{B}$ is projected to its first component). Because τ is accepting, the R- and G-components cannot forever remain $\perp$, so τ contains a transition of type (2), say at time k. Let $j \in \{1, \dots, m\}$ be arbitrary and suppose that $s_p \models R_j(r_p, r_{p+1})$ holds for infinitely many $p \geq k$. Because the R- and G-components are infinitely often reset to $\emptyset$, there must be infinitely many q such that $s_q \models G_j(r_q, r_{q+1})$. Therefore, ρ is accepted by $\mathcal{A}$. Q.E.D.

3.3 Determinization of Edge Automata

The fundamental theorem in the theory of ω-automata asserts that the complement of an ω-regular languages is again ω-regular. The original proof by Büchi [25] is based on combinatorial arguments (Ramsey's theorem) and therefore does not yield an effective complementation procedure. Starting with McNaughton [72], several automata constructions for complementation have been defined. As for classical NFAs, the overall approach is to first construct an equivalent deterministic automaton, which is then easy to complement. Since deterministic Büchi automata are strictly weaker than nondeterministic ones, stronger acceptance conditions, such as Rabin and Streett conditions, are necessary for the deterministic automaton.

Figure 3.2 shows a simple example that illustrates why the standard subset construction that is used to construct an equivalent DFA for a given NFA cannot be used for Büchi automata: the Büchi edge automaton $\mathcal{A}$ of figure 3.2(a) defines the empty language, because the accepting transition (q_0, q_1) cannot be taken during any infinite run. On the other hand, the automaton $\mathcal{B}$ of figure 3.2(b) accepts every behavior. However, the same transition table, shown in figure 3.2(c), is obtained by the standard subset construction for both $\mathcal{A}$ and $\mathcal{B}$, showing that the construction is inadequate for determinizing Büchi automata.

The problem with the subset construction lies with the fact that it only records the set of reachable states. However, the fact that some state may be reachable via an accepting transition for infinitely many prefixes of a behavior does not guarantee that there is an accepting run of the automaton over the entire behavior. In fact, it has been shown [77, 99] that there are Büchi automata with n states whose complement language cannot be defined with an automaton of $2^{O(n)}$ states, even allowing Rabin or Streett acceptance conditions.

We will follow an (essentially) optimal refinement of the subset construction that was defined by Safra [88] and has complexity $2^{O(n \log n)}$. The basic idea is to organize the locations of the deterministic automaton as trees labelled by sets of locations of the original automaton that correspond to different "threads" of runs. Each node evolves according to the usual subset construction, but spawns a new "thread" for those locations that can be reached via an accepting transition. Special measures are taken to bound the breadth and the width of the resulting tree by "merging" threads that need no longer be distinguished.

The accepting runs of the resulting automaton are defined by a Rabin condition. Complementation therefore produces a deterministic Streett automaton, from which a non-deterministic Büchi automaton can be obtained by the construction of theorem 3.2. A different approach to complementation is based on alternating automata [58, 80, 100] and strategies for the games defined by these automata.

We now introduce a variant of Safra's construction that produces a deterministic, loop-free Rabin edge automaton for a given Büchi edge automaton. We will need to refer to the details of this construction in the completeness proof of section 4.6. Safra's construction is based on an auxiliary tree structure:

Definition 3.3 (k-tree) *For any $k \geq 1$ and any set L (of labels), a k-tree (T, λ) over L is a finite tree T, together with a labelling function λ that assigns non-empty subsets of L to the nodes of T such that the following conditions hold:*

K1. The label of each node is a strict superset of the union of the labels of its children.

K2. The labels of sibling nodes are disjoint.

Define a partial ordering $\prec$ on the nodes of a k-tree by $v \prec w$ iff some ancestor of v is an elder sibling of some ancestor of w.

Lemma 3.4 *Assume that (T, λ) is a k-tree over label set L.*

1. *For any nodes v, w of T, if $v \prec w$ then $\lambda(v) \cap \lambda(w) = \emptyset$.*

2. *T has at most $|L|$ nodes.*

Proof. First observe that condition (K1) ensures, inductively, that the label set $\lambda(v)$ of every node v is contained in the label set $\lambda(w)$ of any ancestor w of v.

1. Assume that v, w are nodes of T such that $v \prec w$, and let v' and w' denote the ancestors of v and w such that v' is an elder sibling of w'. Then,

$$\begin{array}{rcll} \lambda(v) \cap \lambda(w) & \subseteq & \lambda(v') \cap \lambda(w') & \text{(by the observation above)} \\ & = & \emptyset & \text{(by condition K2)} \end{array}$$

2. It suffices to prove that for every node v of T we can pick some label $l \in L$ such that $l \in \lambda(w)$ holds only for node v and its ancestors in T. Condition (K1) ensures that there is some $l \in \lambda(v)$ that is not contained in the label of any child of v. The above observation implies that l can not be contained in the label set of any node below v in T. For nodes w that are neither above or below v in T, we have either $v \prec w$ or $w \prec v$. Therefore, assertion (1) ensures $\lambda(v) \cap \lambda(w) = \emptyset$, and in particular $l \notin \lambda(w)$. Q.E.D.

Our variant of Safra's construction is shown in figure 3.3. As an illustration, figure 3.4 shows (the reachable part of) the Rabin automaton that is obtained by applying this construction to the example automaton of figure 3.1 (page 28), choosing $V = \{p\}$. The construction obviously ensures that the resulting automaton is deterministic. Moreover, the locations of the deterministic automaton consist of n-trees that are tagged with an extra bit (shown in the upper right corner of states in figure 3.4), and self-loops are replaced with transitions to a location with the same n-tree, but the opposite tag. This ensures that the resulting automaton is loop-free. The acceptance condition of the example automaton requires that at least one of the edges $(\overline{q_3}, \overline{q_4})$ or $(\overline{q_4}, \overline{q_3})$ is taken infinitely often, while the edges $(\overline{q_3}, \overline{q_0})$ and $(\overline{q_4}, \overline{q_0})$ are followed only finitely often. Obviously, this implies that any behavior accepted by the automaton must satisfy $\Diamond\Box p$. It is also easy to see that every such behavior is accepted. Therefore, the Rabin automaton defines the same language as the original Büchi automaton.

We note the following lemma concerning the numbering of nodes in the Rabin automaton produced by Safra's construction. Here and in the following, we use notation such as $\overline{q}.i$, for a location $\overline{q}$ of $\overline{\mathcal{A}}$ and $i \in \{1, \ldots, n\}$, to refer to the node numbered i in the n-tree of location $\overline{q}$.

Lemma 3.5 *Let $\overline{\mathcal{A}} = (\overline{Q}, \overline{q_0}, \overline{\delta}, \overline{C})$ be the result of applying the construction of figure 3.3 to the Büchi automaton $\mathcal{A} = (Q, q_0, \delta, C)$.*

Input. Büchi edge automaton $\mathcal{A} = (Q, q_0, \delta, C)$ where $At(\delta(q,q')) \subseteq V$ for all $q, q' \in Q$ and $|Q| = n$.

Output. deterministic, loop-free Rabin edge automaton $\overline{\mathcal{A}} = (\overline{Q}, \overline{q_0}, \overline{\delta}, \overline{C})$.

Locations. The locations of $\overline{\mathcal{A}}$ are of the form (T, λ, b) where (T, λ) is an n-tree over Q whose nodes are identified by numbers in $\{1, \ldots, n\}$, and $b \in \{0,1\}$. Besides, $\overline{Q}$ contains two special sink states $q_\perp$ and $\overline{q}_\perp$.

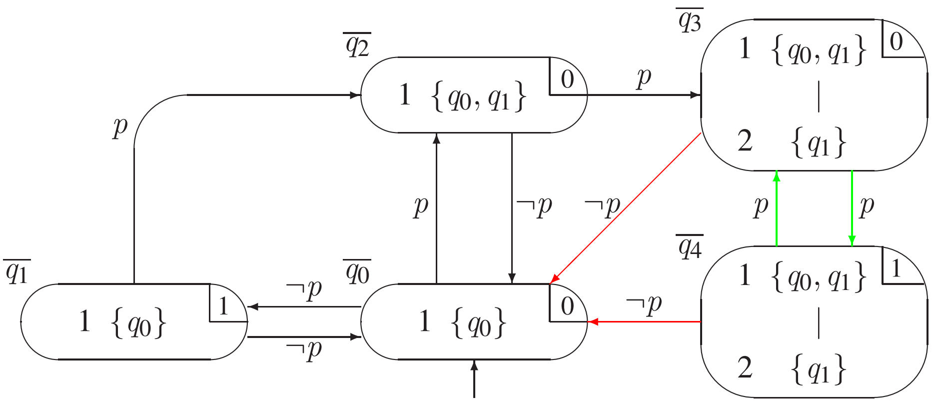
Initial location. $\overline{q_0} = (T, \lambda, 0)$ where T consists of a single node, numbered 1, and labelled by $\{q_0\}$.

Transition conditions. For $s \subseteq V$, let $\chi_s \equiv \bigwedge_{v \in s} v \wedge \bigwedge_{v \in V \setminus s} \neg v$.
Define $\overline{\delta}(q_\perp, \overline{q}_\perp) = \overline{\delta}(\overline{q}_\perp, q_\perp) =$ **true** and $\overline{\delta}(q_\perp, q_\perp) = \overline{\delta}(\overline{q}_\perp, \overline{q}_\perp) =$ **false**. For any other location $\overline{q} = (T, \lambda, c, b)$ and every set $s \subseteq V$, construct a unique successor $succ(\overline{q}, s)$ as follows; the transition condition $\overline{\delta}(\overline{q}, \overline{q}')$ is then defined as the disjunction of all χ_s such that $\overline{q}' = succ(\overline{q}, s)$.

1. Construct a tree of at most $2n$ nodes over Q by making a copy of T, but replacing the label set of each node v by the set of possible δ-successors,
$$\delta(\lambda(v), s) \triangleq \{q' \in Q \ : \ \models \chi_s \rightarrow \delta(q, q') \text{ for some } q \in \lambda(v)\}$$
and creating a new (youngest) child of all nodes v of T, labelled by $\delta(\lambda(v), s) \cap C$. Let this new child be numbered by $i + n$, where i is the number of v in T.
2. For all nodes $v \prec w$, delete all Q-locations $q \in \lambda(v)$ from $\lambda(w)$.
3. Remove all nodes with empty labels. If the root is removed, $succ(\overline{q}, s) = q_\perp$.
4. Remove the descendants of all nodes whose label set equals the union of the label sets of its children.
5. The tree now satisfies (K1) and (K2), and therefore has at most n nodes. Renumber the nodes such that all nodes with numbers $\leq n$ keep their number. Let (T', λ') be the resulting n-tree. Define $\overline{q}' = succ(\overline{q}, s) = (T', \lambda', b')$ where $b' = 1 - b$ if $(T', \lambda') = (T, \lambda)$, and $b' = 0$ otherwise. For $i = 1, \ldots, n$, define the colors $c_i^s(\overline{q}, \overline{q}')$ by
$$c_i^s(\overline{q}, \overline{q}') = \begin{cases} \text{red} & \text{if node } i \text{ was deleted in step 3} \\ \text{green} & \text{if step 4 was applied to node } i \\ \text{white} & \text{otherwise} \end{cases}$$

Acceptance conditions. $\overline{C} = \{(R_1, G_1), \ldots, (R_n, G_n)\}$ where the formulas $R_i(\overline{q}, \overline{q}')$ and $G_i(\overline{q}, \overline{q}')$ are the disjunctions of all χ_s such that $\overline{q}' \notin \{q_\perp, \overline{q}_\perp\}$ is the successor of $\overline{q}$ and s and $c_i^s(\overline{q}, \overline{q}') =$ red (resp., $c_i^s(\overline{q}, \overline{q}') =$ green).

Figure 3.3: Determinization of edge automata.



$$
\begin{array}{rcll}
R_1(\overline{q},\overline{q}') & \equiv & \textbf{false} & \\
R_2(\overline{q_3},\overline{q_0}) & \equiv & \neg p & \\
R_2(\overline{q_4},\overline{q_0}) & \equiv & \neg p & \\
R_2(\overline{q},\overline{q}') & \equiv & \textbf{false} & \text{otherwise}
\end{array}
\qquad
\begin{array}{rcll}
G_1(\overline{q},\overline{q}') & \equiv & \textbf{false} & \\
G_2(\overline{q_3},\overline{q_4}) & \equiv & p & \\
G_2(\overline{q_4},\overline{q_3}) & \equiv & p & \\
G_2(\overline{q},\overline{q}') & \equiv & \textbf{false} & \text{otherwise}
\end{array}
$$

Figure 3.4: Determinizing the automaton of figure 3.1.

1. *For any locations $\overline{q},\overline{q}' \in \overline{Q} \setminus \{q_\perp, \overline{q}_\perp\}$, any $s \subseteq V$, and any $i \in \{1,\ldots,n\}$, if $\overline{q}'$ is the successor of $\overline{q}$ and s, if $\overline{q}'.i$ exists, and $c_i^s(\overline{q},\overline{q}') \neq red$ then $\overline{q}.i$ also exists, and for every $\mathcal{A}$-location q' in the label of $\overline{q}'.i$ there exists some $\mathcal{A}$-location q in the label of $\overline{q}.i$ such that $\models \chi_s \rightarrow \delta(q,q')$.*

2. *If $\overline{q}_1 \xrightarrow{s_1} \overline{q}_2 \xrightarrow{s_2} \ldots \xrightarrow{s_m} \overline{q}_{m+1}$ is a path in $\overline{\mathcal{A}}$ where $\overline{q}_i \notin \{q_\perp, \overline{q}_\perp\}$ for all $i = 1,\ldots,m+1$ and such that $s_1 \models G_i(\overline{q}_1,\overline{q}_2)$ but $s_k \not\models R_i(\overline{q}_k,\overline{q}_{k+1})$ and $s_k \not\models G_i(\overline{q}_k,\overline{q}_{k+1})$ for all $k = 2,\ldots,m$, then $\overline{q}_{m+1}.i$ exists and for every child w of $\overline{q}_{m+1}.i$ and every $\mathcal{A}$-location q' in the label of w there exists some q in the label of $\overline{q}_2.i$ such that $q \overset{s_2\ldots s_m}{\Longrightarrow}_C q'$ is a path in $\mathcal{A}$.*

3. *If $\overline{q}_1 \xrightarrow{s_1} \overline{q}_2 \xrightarrow{s_2} \ldots \xrightarrow{s_m} \overline{q}_{m+1} \xrightarrow{s_{m+1}} \overline{q}_{m+2}$ is a path in $\overline{\mathcal{A}}$ as in (2) but where additionally $s_{m+1} \models G_i(\overline{q}_{m+1},\overline{q}_{m+2})$ then for every $\mathcal{A}$-location q' in the label of $\overline{q}_{m+2}.i$ there exists some q in the label of $\overline{q}_2.i$ such that $q \overset{s_2\ldots s_{m+2}}{\Longrightarrow}_C q'$ is a path in $\mathcal{A}$.*

Proof. Assertion 1 follows immediately from the definition of $\overline{\mathcal{A}}$ and the rules for coloring edges. Assertion 2 is proved by induction on $m \geq 1$, noting that $\overline{q}_{m+1}.i$ has no children, and that children are created only when there are successors via accepting transitions in the original automaton $\mathcal{A}$. Finally, assertion 3 follows from (2) and the rules to color a transition green. Q.E.D.

We are now in a position to prove the correctness of Safra's construction.

Theorem 3.6 *For any Büchi edge automaton $\mathcal{A}$, the construction of figure 3.3 produces an equivalent loop-free and deterministic Rabin edge automaton $\overline{\mathcal{A}}$.*

Proof. We have already argued that $\overline{\mathcal{A}}$ is deterministic and loop-free, so it only remains to prove that $\mathcal{L}(\overline{\mathcal{A}}) = \mathcal{L}(\mathcal{A})$.

To show the equivalence of the automata $\mathcal{A}$ and $\overline{\mathcal{A}}$, first assume that some behavior $\sigma = s_0 s_1 \ldots$ is accepted by $\overline{\mathcal{A}}$. Let $\rho = r_0 r_1 \ldots$ be the (unique) run of $\overline{\mathcal{A}}$ over σ, and let $i \in \{1, \ldots, n\}$ be such that $s_k \models R_i(r_k, r_{k+1})$ holds for only finitely many $k \in \mathbb{N}$ but $s_k \models G_i(r_k, r_{k+1})$ for infinitely many $k \in \mathbb{N}$. In particular, the definition of G_i implies that $r_k \notin \{q_\perp, \overline{q}_\perp\}$ for all $k \in \mathbb{N}$. Choose j_0 such that $s_{j_0} \models G_i(r_{j_0}, r_{j_0+1})$ and $s_k \not\models R_i(r_k, r_{k+1})$ for all $k \geq j_0$. Furthermore, denote by $j_0 < j_1 < j_2 < \ldots$ all positions beyond j_0 such that $s_{j_k} \models G_i(r_{j_k}, r_{j_k+1})$, also let $j_{-1} = 0$. We construct a tree with nodes of the form (q, k) for $\mathcal{A}$-locations q as follows: the root of the tree is $(q_0, -1)$, its successors are all nodes $(q, 0)$ such that q is contained in the label of $r_{j_0}.i$. Inductively, the successors of a node (q, k) are all nodes $(q', k+1)$ such that q' occurs in the label of $r_{j_{k+1}}.i$ and $q \overset{s_{j_k+1} \ldots s_{j_{k+1}}}{\Longrightarrow_C} q'$ is a path in $\mathcal{A}$. From the choice of j_0 and lemma 3.5:1 it follows that $r_k.i$ exists for all $r \geq j_0$, and lemma 3.5:3 implies that the tree defined above has infinitely many nodes. On the other hand, the width of the tree is bounded by $|Q| = n$, and therefore König's lemma implies that the tree contains an infinite path $\pi = (q_0, -1)(p_0, 0)(p_1, 1) \ldots$. By construction, every edge $((p_k, k), (p_{k+1}, k+1))$ of π (for $k \geq 0$) corresponds to an $\mathcal{A}$-path $p_k \overset{s_{j_k+1} \ldots s_{j_{k+1}}}{\Longrightarrow_C} p_{k+1}$ that contains an accepting $\mathcal{A}$-transition. Moreover, it cannot be the case that $s_k \models R_1(r_k, r_{k+1})$ for any $k \geq 0$, for then the label of the root (which is always numbered 1) would have become empty, and no subsequent location would contain a node numbered i either. Condition (K1) of definition 3.3 ensures that p_0 is contained in the label of the root of $\overline{\mathcal{A}}$-location r_{j_0}, and lemma 3.5:2 implies the existence of an $\mathcal{A}$-path $q_0 \overset{s_0 \ldots s_{j_0}}{\Longrightarrow} p_0$ from q_0, the only element in the label of the root of location r_0, to p_0. The concatenation of all these finite paths yields a run of $\mathcal{A}$ over σ that contains infinitely many accepting transitions and is therefore accepting.

On the other hand, assume that σ is accepted by $\mathcal{A}$, via accepting run $\pi = q_0 q_1 \ldots$; we have to prove that the run $\rho = \overline{q}_0 \overline{q}_1 \ldots$ of $\overline{\mathcal{A}}$ over σ is accepting. The construction of $\overline{\mathcal{A}}$ ensures that the $\mathcal{A}$-location q_k is always contained in the label of the root of $\overline{\mathcal{A}}$-location $\overline{q_k}$; in particular, the root is never removed in step 3, and $\overline{q}_k \in \{q_\perp, \overline{q}_\perp\}$ or $s_k \models R_1(\overline{q}_k, \overline{q}_{k+1})$ holds for no $k \in \mathbb{N}$. If $s_k \models G_1(\overline{q}_k, \overline{q}_{k+1})$ holds for infinitely many $k \in \mathbb{N}$, we are done. Otherwise, there exists some smallest $j \in \mathbb{N}$ such that $s_k \models G_1(\overline{q}_k, \overline{q}_{k+1})$ for no $k \geq j$. Let $l \geq j$ be the index of the first accepting $\mathcal{A}$-transition beyond j, i.e. $s_l \models C(q_l, q_{l+1})$; note that π contains infinitely many accepting transitions. The construction of $\overline{\mathcal{A}}$ (step 1) implies that q_{l+1} is contained in the label of some child of the root in location $\overline{q}_{l+1}$. For $k > l$, the $\mathcal{A}$-location q_k will be contained in the label of some child of the root. It may be moved to some older child by step 2, but since an n-tree has only finitely many nodes, there exists some $m \geq l$ and some child of the root, say, numbered c, whose label contains q_k for all $k \geq m$. In particular, this child is never removed beyond position m, so $s_k \models R_c(\overline{q}_k, \overline{q}_{k+1})$ holds for at most finitely many $k \in \mathbb{N}$. Either $s_k \models G_c(\overline{q}_k, \overline{q}_{k+1})$ holds for infinitely many $k \in \mathbb{N}$, and we are done, or we repeat the argument for the children of c. Because the depth of an n-tree is bounded, we must eventually reach some index for which ρ contains infinitely many "green", but only finitely many "red" transitions, proving that ρ is an accepting run of $\overline{\mathcal{A}}$ over σ. Q.E.D.

3.4 Characterizing Stuttering Invariance

We have seen in theorem 2.5 that quantified TLA* formulas are insensitive to stuttering equivalence. We will now introduce a subclass of Büchi edge automata whose transition and acceptance conditions enjoy certain "transitivity" properties. This class of automata will be shown to characterize precisely the class of ω-regular languages that are invariant under stuttering equivalence.

Definition 3.7 (SI$_V$ automaton) *Let $V \subseteq \mathcal{V}$ be some set of atomic propositions. A Büchi edge automaton $\mathcal{A} = (Q, q_0, \delta, C)$ is an* SI$_V$ automaton *if the following conditions hold for all $q, q', q'' \in Q$:*

SI0. $At(\delta(q,q')) \subseteq V$ *and* $At(C(q,q')) \subseteq V$

SI1. $\models \delta(q,q') \rightarrow \delta(q',q')$

SI2. $\models \delta(q,q') \wedge \delta(q',q'') \rightarrow \delta(q,q'')$

SI3. (a) $\models \delta(q,q') \wedge \delta(q',q'') \wedge C(q,q') \rightarrow C(q,q'')$
(b) $\models \delta(q,q') \wedge \delta(q',q'') \wedge C(q',q'') \rightarrow C(q,q'')$

Condition (SI0) requires that only propositions from V appear in transition and acceptance conditions of $\mathcal{A}$; this implies that from any location, the same set of transitions can be taken for any two V-similar states. Condition (SI1) ensures that the automaton can remain in the target state of a transition for as long as states are repeated (up to V-similarity). On the other hand, condition (SI2) requires the transition relation to be "transitive" in the sense that whenever both transitions $q \xrightarrow{s} q'$ and $q' \xrightarrow{t} q''$ are possible for V-similar states s and t then the transition $q \xrightarrow{s} q''$ can be taken in a single step. Finally, (SI3) stipulates that in any such case, if one of the transitions (q,q') or (q',q'') is accepting then so is the direct transition (q,q'').

For example, the automaton of figure 3.1 is an SI$_{\{p\}}$ automaton. Obviously, if $\mathcal{A}$ is an SI$_V$ automaton and $V \subseteq W$ then $\mathcal{A}$ is also an SI$_W$ automaton.

Lemma 3.8 *Assume that $\mathcal{A} = (Q, q_0, \delta, C)$ is an SI$_V$ automaton and that s and $s_1, \ldots, s_n$ are states such that $s \simeq_V s_i$ holds for all $i \in \{1, \ldots, n\}$ (where $n \geq 1$). Then for all $q, q' \in Q$:*

1. $q \overset{s_1 \ldots s_n}{\Longrightarrow} q'$ *if and only if* $q \xrightarrow{s} q'$.

2. $q \overset{s_1 \ldots s_n}{\Longrightarrow}_C q'$ *if and only if* $s \models \delta(q,q') \wedge C(q,q')$.

Proof. The proof is by induction on $n \geq 1$.

For $n = 1$, assertion (1) requires to show that $s \models \delta(q,q')$ iff $s_1 \models \delta(q,q')$, which follows from condition (SI0) because s and s_1 agree on all propositions in V. The proof of assertion (2) is similar.

For the induction step, assume that $q \overset{s_1 \ldots s_{n+1}}{\Longrightarrow} q'$. So there exist locations $q_1, q_2, \ldots, q_{n+2}$ where $q = q_1$ and $q' = q_{n+2}$ such that $s_i \models \delta(q_i, q_{i+1})$ for all $i \in \{1, \ldots, n+1\}$. From the induction hypothesis, we obtain $s \models \delta(q_1, q_{n+1})$. Moreover, since $s_{n+1} \models \delta(q_{n+1}, q_{n+2})$ and $s \simeq_V s_{n+1}$, condition (SI0) implies that $s \models \delta(q_{n+1}, q_{n+2})$. By condition (SI2) it follows that $s \models \delta(q_1, q_{n+2})$, and therefore $q \xrightarrow{s} q'$ as required.

On the other hand, assume that $s \models \delta(q,q')$. Condition (SI1) implies that $s \models \delta(q',q')$, and by (SI0) we obtain $s_1 \models \delta(q,q')$ and $s_i \models \delta(q',q')$ for $i = 2, \ldots, n+1$. This proves $q \overset{s_1 \ldots s_{n+1}}{\Longrightarrow} q'$.

The proof of assertion (2) is similar, relying also on condition (SI3). Q.E.D.

Lemma 3.8 is the key to proving that the language defined by an SI_V automaton is closed under V-stuttering equivalence.

Theorem 3.9 *Let $\mathcal{A} = (Q, q_0, \delta, C)$ be an SI_V automaton and σ, τ be behaviors such that $\sigma \simeq_V \tau$. Then $\sigma \in \mathcal{L}(\mathcal{A})$ iff $\tau \in \mathcal{L}(\mathcal{A})$.*

Proof. It is enough to prove that $\sigma \in \mathcal{L}(\mathcal{A})$ iff $\natural_V \sigma \in \mathcal{L}(\mathcal{A})$, for an arbitrary V-stuttering free variant $\natural_V \sigma$ of σ. So assume that $\sigma = s_0 s_1 \ldots$, that $\natural_V \sigma = t_0 t_1 \ldots$, and that σ can be represented in the form $\sigma = \langle s_{j_0} \ldots s_{j_1 - 1} \rangle \circ \langle s_{j_1} \ldots s_{j_2 - 1} \rangle \circ \cdots$ where $0 = j_0 < j_1 < \ldots$ and $s_k =_V t_i$ for all $i \in \mathbb{N}$ and all $k = j_i, \ldots, j_{i+1} - 1$. Now,

$\sigma \in \mathcal{L}(\mathcal{A})$

iff there exist $q_1, q_2, \ldots$ such that
$q_i \overset{s_{j_i} \ldots s_{j_{i+1}-1}}{\Longrightarrow} q_{i+1}$ for all $i \in \mathbb{N}$ and
$q_i \overset{s_{j_i} \ldots s_{j_{i+1}-1}}{\Longrightarrow}_C q_{i+1}$ for infinitely many $i \in \mathbb{N}$

iff there exist $q_1, q_2, \ldots$ such that (by proposition 3.8)
$q_i \xrightarrow{s_i} q_{i+1}$ for all $i \in \mathbb{N}$ and
$s_i \models C(q_i, q_{i+1})$ for infinitely many $i \in \mathbb{N}$

iff $\natural_V \sigma \in \mathcal{L}(\mathcal{A})$. Q.E.D.

On the other hand, we show that for every Büchi automaton whose transition formulas are built from propositions in the finite set $V \subseteq \mathcal{V}$ we can construct an SI_V automaton $\mathcal{A}_{sc}^V$ that defines precisely the V-stuttering closure of $\mathcal{L}(\mathcal{A})$. In a sense, this implies that SI_V automata characterize precisely the set of ω-regular languages closed under V-stuttering equivalence, for "sufficiently large" V. The construction of $\mathcal{A}_{sc}$ appears in figure 3.5. As in figure 3.3, the formula χ_s (for

Input. Büchi edge automaton $\mathcal{A} = (Q, q_0, \delta, C)$ where
$At(\delta(q,q')) \cup At(C(q,q')) \subseteq V$ for all $q, q' \in Q$

Output. SI_V automaton $\mathcal{A}_{sc}^V = (\overline{Q}, q_0, \overline{\delta}, \overline{C})$

Locations. $\overline{Q} = \{q_0\} \cup (Q \times 2^V)$

Transition conditions.

$$\begin{aligned}
\overline{\delta}(q_0,(q,s)) &\equiv \begin{cases} \chi_s & \text{if } q_0 \overset{s^k}{\Rightarrow} q \text{ for some } k \geq 1 \\ \textbf{false} & \text{otherwise} \end{cases} \\
\overline{\delta}((q',s'),(q,s)) &\equiv \begin{cases} \chi_s & \text{if } q' \overset{s^k}{\Rightarrow} q \text{ for some } k \geq 1 \\ & \quad \text{or } (q',s') = (q,s) \\ \textbf{false} & \text{otherwise} \end{cases} \\
\overline{\delta}(\overline{q},q_0) &\equiv \textbf{false} \quad \text{for all } \overline{q} \in \overline{Q}
\end{aligned}$$

Acceptance conditions.

$$\begin{aligned}
\overline{C}((q',s'),(q,s)) &\equiv \begin{cases} \chi_s & \text{if } q' \overset{s^k}{\Rightarrow}_C q \text{ for some } k \geq 1 \\ \textbf{false} & \text{otherwise} \end{cases} \\
\overline{C}(q_0,\overline{q}) \equiv \overline{C}(\overline{q},q_0) &\equiv \textbf{false} \quad \text{for all } \overline{q} \in \overline{Q}
\end{aligned}$$

Figure 3.5: Construction of $\mathcal{A}_{sc}^V$.

$s \subseteq V$) denotes the formula $\bigwedge_{v \in s} v \wedge \bigwedge_{v \in V \setminus s} \neg v$. For succinctness, we have made some abuse of notation and have identified a set $s \subseteq V$ of atomic propositions with the state that satisfies precisely the propositions in s.

Lemma 3.10 *Assume that $\mathcal{A} = (Q, q_0, \delta, C)$ is a Büchi edge automaton where $At(\delta(q,q')) \cup At(C(q,q')) \subseteq V$ for all $q, q' \in Q$. Then the automaton $\mathcal{A}_{sc}^V$ as defined in figure 3.5 is an SI_V automaton.*

Proof. Condition (SI0) is obviously satisfied because all transition and acceptance conditions are either **false** or of the form χ_s for some $s \subseteq V$, implying $At(\chi_s) = V$.

To prove condition (SI1), first observe that $\overline{\delta}(\overline{q}, q_0) \equiv$ **false** for all $\overline{q} \in \overline{Q}$, hence

$$\models \overline{\delta}(\overline{q}, q_0) \rightarrow \overline{\delta}(q_0, q_0)$$

holds trivially. For $\overline{q}' = (q', s')$, we have $\overline{\delta}(\overline{q}', \overline{q}') \equiv \chi_{s'}$, whereas $\overline{\delta}(\overline{q}, \overline{q}')$ is either **false** or $\chi_{s'}$ for any $\overline{q} \in \overline{Q}$, and

$$\models \overline{\delta}(\overline{q}, \overline{q}') \rightarrow \overline{\delta}(\overline{q}', \overline{q}')$$

holds in either case.

For condition (SI2), $\overline{\delta}(\overline{q},\overline{q}') \wedge \overline{\delta}(\overline{q}',\overline{q}'')$ is different from **false** only if $\overline{q}'$ and $\overline{q}''$ are of the form $\overline{q}' = (q',s)$ and $\overline{q}'' = (q'',s)$ for some locations $q',q'' \in Q$ and some $s \subseteq V$, if $q' = q''$ or $q' \stackrel{s^k}{\Rightarrow} q''$ and additionally q' is reached by a (possibly empty) sequence s^l from the $\mathcal{A}$-location underlying $\overline{q}$, in which case all transition conditions $\overline{\delta}(\overline{q},\overline{q}')$, $\overline{\delta}(\overline{q}',\overline{q}'')$, and $\overline{\delta}(\overline{q},\overline{q}'')$ equal χ_s, thus implying the assertion.

The proof of (SI3) is similar. Q.E.D.

The following theorem asserts the correctness of the construction of $\mathcal{A}_{sc}^V$.

Theorem 3.11 *Let $\mathcal{A}$ and $\mathcal{A}_{sc}^V$ be Büchi edge automata as in figure 3.5. Then $\mathcal{L}(\mathcal{A}_{sc}^V) = \mathcal{SC}_V(\mathcal{L}(\mathcal{A}))$.*

Proof. By lemma 3.10, $\mathcal{A}_{sc}^V$ is an SI_V automaton, and therefore theorem 3.9 implies that $\mathcal{L}(\mathcal{A}_{sc}^V)$ is closed under V-stuttering equivalence. It is therefore enough to show that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}_{sc}^V)$ and that for every $\sigma \in \mathcal{L}(\mathcal{A}_{sc}^V)$ there exists some $\tau \in \mathcal{L}(\mathcal{A})$ such that $\sigma \simeq_V \tau$.

Assume first that $\sigma \in \mathcal{L}(\mathcal{A})$ for $\sigma = s_0 s_1 \ldots$ and that $\rho = r_0 r_1 \ldots$ is an accepting run of $\mathcal{A}$ over σ. Let t_i denote $s_i \cap V$, i.e. the restriction of s_i to the atomic propositions in V (again identifying a state with the set of atomic propositions true of the state); note that the assumption that $At(\delta(q,q')) \subseteq V$ for all $q,q' \in Q$ implies that $s_i \models \delta(q,q')$ iff $t_i \models \delta(q,q')$, and similarly for the acceptance conditions.

Now let $\overline{r}_0 = r_0$ and $\overline{r}_i = (r_i, t_{i-1})$ for $i \geq 1$. The definition of $\mathcal{A}_{sc}^V$ immediately implies that $\overline{r}_0\overline{r}_1 \ldots$ is an accepting run of $\mathcal{A}_{sc}^V$ over σ. This proves that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}_{sc}^V)$.

Assume now that $\sigma \in \mathcal{L}(\mathcal{A}_{sc}^V)$ for $\sigma = s_0 s_1 \ldots$ and that $\overline{\rho} = \overline{r}_0\overline{r}_1 \ldots$ is an accepting run of $\mathcal{A}_{sc}^V$ over σ. Again, we write t_i for $s_i \cap V$. The definition of $\mathcal{A}_{sc}^V$ implies that $\overline{r}_0 = q_0$ and that, for each $i \geq 1$, $\overline{r}_i$ is of the form (q_i, t_{i-1}) for some $q_i \in Q$. We inductively define finite behaviors τ_i and finite runs ρ_i of $\mathcal{A}$ over τ_i. Let $\tau_0 = \varepsilon$ and $\rho_0 = q_0$. Inductively, assume that τ_m and ρ_m have been defined and consider the transition $\overline{r}_m \xrightarrow{s_m} \overline{r}_{m+1}$ in $\overline{\rho}$. We distinguish two cases:

1. If $s_m \models \overline{C}(\overline{r}_m,\overline{r}_{m+1})$ then choose some $k \geq 1$ such that $q_m \stackrel{s_m^k}{\Rightarrow}_C q_{m+1}$ holds for $\mathcal{A}$ (such k exists by the definition of $\overline{C}$), let $\tau_{m+1} = \tau_m s_m^k$ and $\rho_{m+1} = \rho_m x$ where x is the sequence of locations contained in the above s_m^k-path from q_m to q_{m+1}, without the initial q_m. Observe that $|x| = k \geq 1$, and that the path $q_m x$ contains at least one accepting transition.

2. Otherwise choose some $k \geq 0$ such that $q_m \stackrel{s_m^k}{\Rightarrow} q_{m+1}$ (such k exists by the definition of $\overline{\delta}$), let $\tau_{m+1} \stackrel{\Delta}{=} \tau_m s_m^k$ and $\rho_{m+1} \stackrel{\Delta}{=} \rho_m x$, where x is defined as above. (Here x may be ε if $k = 0$, which implies that $q_m = q_{m+1}$ and $t_{m-1} = t_m$, thus $s_{m-1} =_V s_m$; this case corresponds to removal of V-stuttering.)

The definition ensures that $\tau_m \simeq_V s_0 \ldots s_{m-1}$ holds for all $m \geq 0$. Also note that $\overline{\rho}$ is an accepting run of $\mathcal{A}_{sc}^V$ over σ, hence case 1 must occur infinitely often, and therefore the limit τ of the behaviors $\tau_0, \tau_1, \ldots$ is infinite and V-stuttering equivalent to the original behavior σ. Moreover, the limit ρ of $\rho_0, \rho_1, \ldots$ is an infinite run of $\mathcal{A}$ over τ and contains infinitely many accepting transitions. Therefore we have $\tau \in \mathcal{L}(\mathcal{A})$, which completes the proof. Q.E.D.

Theorems 3.9 and 3.11 imply that ω-regular languages that are closed under stuttering can be defined by SI_V automata, for "large enough" V.

Corollary 3.12 *An ω-regular language $\mathcal{L}$ of behaviors is closed under stuttering iff $\mathcal{L} = \mathcal{L}(\mathcal{A})$ for some SI_V automaton $\mathcal{A}$, for some finite set $V \subseteq \mathcal{V}$ of atomic propositions.*

Proof. Lemma 3.9 ensures that the language defined by an SI_V automaton is closed under V-stuttering equivalence, and therefore also under stuttering equivalence. Conversely, if $\mathcal{L}$ is an ω-regular language that is closed under stuttering equivalence, let $\mathcal{B} = (Q, q_0, \delta, C)$ be some Büchi edge automaton such that $\mathcal{L}(\mathcal{B}) = \mathcal{L}$, and let

$$V = \bigcup_{q,q' \in Q} At(\delta(q,q')) \cup At(C(q,q'))$$

Because $\mathcal{B}$ cannot distinguish between V-similar behaviors and $\mathcal{L}(\mathcal{B})$ is closed under stuttering equivalence, it follows that $\mathcal{B}$ is even closed under V-stuttering equivalence, that is, $\mathcal{SC}_V(\mathcal{L}(\mathcal{B})) = \mathcal{L}(\mathcal{B})$. Now, by lemma 3.10, we know that $\mathcal{B}_{sc}^V$ is an SI_V automaton, and theorem 3.11 implies that $\mathcal{L}(\mathcal{B}_{sc}^V) = \mathcal{SC}_V(\mathcal{L}(\mathcal{B})) = \mathcal{L}(\mathcal{B}) = \mathcal{L}$. Q.E.D.

Chapter 4

Quantified TLA*

4.1 Overview

We will now study an extension of the logic introduced in chapter 2 by quantification over propositions. In particular, we focus on proof rules for quantified pTLA* formulas. When writing system specifications, existential quantification arises when internal, auxiliary state components are hidden from the externally visible interface. Typically, TLA specifications are of the form

$$\exists\, x : \mathit{Init} \wedge \Box[\mathit{Next}]_v \wedge L$$

where x is a list of internal variables, Init is a state predicate that describes the initial states, Next characterizes the system's next-state relation, and L states the liveness conditions expected from the system. This formula essentially describes a state machine, augmented by fairness and liveness conditions, that generates the allowed behaviors of the system under specification. For such a model-oriented specification style to yield a suitably abstract description, it is of crucial importance to hide the internal state of the specification, and therefore quantification is of critical importance.

During system development, the initial specification is later refined by "concrete" specifications of the same general form as above, but typically more detailed descriptions of the system state and actions are introduced. The concept of stuttering invariance allows (behavioral) refinement to be expressed as implication. The refinement proof requires to construct a "witness" for the internal state of the state machine of the abstract specification to prove that the implementation behaves as allowed by the original specification. This corresponds to the usual introduction rule for proving formulas $F \rightarrow \exists\, x : G$; the computer science literature often calls such witnesses *refinement mappings* [2]. However, the definition of a refinement mapping may require to refer not just to the current values of the implementation's state, but also to past or future values. As a simple example, consider the specification of a compute server that inputs some values i from a client, performs some complex computation $o = f(i)$, and returns the result o. The specification might be written such that the server stores all pairs (i, o) that have already been computed in internal memory, and immediately returns the stored result when receiving some known input. On the other hand, the implementation may decide to replace that unbounded store by a fixed-size cache. Although obviously a correct implementation, it is impossible to define a refinement mapping that would reconstruct the original, unbounded store from the implementation's cache.

It has been proposed to overcome this problem by allowing inductive definitions to be used in the definition of witness terms. We will study syntactic formulations of such definitions in the language of TLA*, and give introduction rules for "history" and "prophecy" variables. As an additional complication, the semantics of quantification in TLA* must allow for the addition or removal of stuttering steps in order to preserve stuttering invariance. For example, the formula

$$\exists\, x : \Box\Diamond\langle \mathbf{true} \rangle_x$$

asserts that it is possible for an internal variable x to change value infinitely often, independently of the externally visible behavior. This formula is valid, but its validity cannot be proven using a refinement mapping. We therefore propose a third kind of auxiliary variables called "stuttering" variables that also allow for a "stretching" of the externally visible transitions.

The main contribution of this chapter is a completeness proof for quantified propositional TLA*. This proof makes heavy use of the concept of edge automata introduced in chapter 3. As a by-product of the completeness proof, we will prove that quantified pTLA* formulas define precisely the class of ω-regular languages that are invariant under stuttering equivalence. We will then define full first-order TLA* and indicate the necessary modifications of our proof rules.

4.2 Quantified propositional TLA*

We now formally define the extension of TLA* by quantification over proposition variables.

Definition 4.1 (syntax of qpTLA***)** *The formulas and pre-formulas of* qpTLA* *are given inductively as in definition 2.1, except by adding the following clause:*

6. *If F is a formula and $x \in \mathcal{V}$ is an atomic proposition then $\exists\, x : F$ is a formula.*

We use standard conventions regarding free and bound propositions in formulas and pre-formulas. In particular, we define the set $At(F)$, for $F \equiv \boldsymbol{\exists}\, v : G$, to be $At(G) \setminus \{v\}$, since v becomes bound by the quantifier. If F, G are formulas and x is a proposition then $F[G/x]$ denotes the formula obtained from F by replacing every free occurrence of x in F by the formula G. If any propositions free in G would become bound by this replacement, we silently assume that bound propositions in F are suitably renamed to avoid such capture of free propositions.

Informally, the formula $\boldsymbol{\exists}\, x : F$ is true of behavior σ if F is true for some behavior τ that differs from σ by the values assigned to x. (The precise semantics is given in definition 4.4 below.) In other words, quantification is over an ω-*sequence* of values rather than over single values. This explains why quantified propositional temporal logics are an interesting object of study: the quantifier $\boldsymbol{\exists}$ can not simply be defined as an abbreviation, as in classical propositional logic. However, it will also be convenient to introduce quantification over values. We thus define (for both formulas and preformulas)

$$\exists x : F \;\equiv\; F[\mathbf{true}/x] \vee F[\mathbf{false}/x]$$

using different symbols to distinguish (rigid) quantification over values from (flexible) quantification. In either case, universal quantification is defined in the standard way as the dual of existential quantification:

$$\boldsymbol{\forall}\, x : F \;\equiv\; \neg\boldsymbol{\exists}\, x : \neg F \qquad \text{and} \qquad \forall x : F \;\equiv\; \neg\exists x : \neg F$$

Before we formally define the semantics of qpTLA*, we first show that the obvious definition would not preserve invariance under stuttering equivalence, and is therefore not appropriate: Consider the formula

$$F \;\equiv\; \boldsymbol{\exists}\, x : \neg x \wedge \Box[x]_v$$

Formula F asserts the existence of some proposition x that is false initially but true whenever v changes its value. In particular, x must change from false to true before v ever changes its value.

Clearly, F holds of the behavior σ represented above because x can be made false in the initial state and true from the second state onwards. However, it would seem that F does not hold of the behavior τ because v changes value during the transition between the first two states, and x would have to be both true and false at the initial state.

On the other hand, the behaviors σ and τ are $\{v\}$-stuttering equivalent, and should therefore either both satisfy or both falsify formula F, whose only free variable is v. Following Lamport [62], we will therefore take care to define the semantics of quantification so that preservation of stuttering invariance is guaranteed.

Definition 4.2 *For $x \in \mathcal{V}$ we define the relations $=_x$ and $\approx_x$ on behaviors as follows:*

1. *Two behaviors $\sigma = s_0 s_1 \ldots$ and $\tau = t_0 t_1 \ldots$ are* equal up to x*, written $\sigma =_x \tau$ if $s_i(v) = t_i(v)$ for all $i \geq 0$ and $v \in \mathcal{V}$, except possibly x.*

2. *The relation $\approx_x$, called* similarity up to x*, is defined as $\approx_x = (\simeq \circ =_x \circ \simeq)$, where $\simeq$ is stuttering equivalence and $\circ$ denotes relational composition.*

In other words, the relation $\approx_x$ allows additional "stretching" of the given behavior to occur before and after modifying the valuation of v. It can be shown that for the purpose of defining the semantics of quantification it would be enough to define $\approx_x = (\simeq \circ =_x)$, that is, to allow stretching only before changing the value of v, because this allows for the insertion of stuttering transitions as necessary for the internal variables to change value (cf. the example above). The second occurrence of $\simeq$ is unnecessary because all formulas are invariant under stuttering equivalence, and therefore additional "stretching" does not change the semantics of the formula. Nevertheless, we prefer to work with definition 4.2 because its symmetry ensures that $\approx_v$ is an equivalence relation, as asserted by the following proposition.

Proposition 4.3

1. *For any $x \in \mathcal{V}$, the relations $=_x$ and $\approx_x$ are equivalence relations.*

2. $(\simeq_V \circ \approx_x) = (\approx_x \circ \simeq_{V \cup \{x\}})$*, for any $x \in \mathcal{V}$ and $V \subseteq \mathcal{V}$.*

The semantics of quantification is now defined in terms of $\approx_x$ instead of $=_x$ in order to ensure invariance under stuttering.

Definition 4.4 (semantics of qpTLA*) *The semantics of* qpTLA* *is obtained by adding the following clause to definition 2.2.*

$$\sigma \models \exists x : F \quad \textit{iff} \quad \tau \models F \textit{ holds for some } \tau \approx_x \sigma$$

Observe that we do not allow quantification over (proper) pre-formulas. For example, $\exists\, x : \circ F$ is not a pre-formula, although $\circ\exists\, x : F$ is. (A similar restriction was imposed on the $\Box$ operator in definition 2.1.) This is because there appears to be no satisfactory semantics definition of quantified pre-formulas: on the one hand, pre-formulas may observe the first two states of a behavior, and therefore the semantics of quantification should not allow to replicate the first state of a behavior before changing the valuation of the bound proposition; the example above has shown that this can be essential to define a stuttering-invariant semantics of quantified formulas. On the other hand, we would like the relations $\models$ and $\models\!\!\!\approx$ to agree on formulas—in particular, we want to preserve soundness of the rule (pre) of Σ_{TLA^*}.

Theorem 4.5 *For any* qpTLA* *formula F and behaviors σ, τ where $\sigma \simeq_{At(F)} \tau$,*

$$\sigma \models F \quad \textit{iff} \quad \tau \models F$$

Proof. Extending the proof of theorem 2.5, we need only consider the case of a quantified formula $F \equiv \exists\, x : G$. So assume that $\sigma \models F$ and that $\tau \simeq_{At(F)} \sigma$. Choose some behavior $\rho \approx_x \sigma$ such that $\rho \models G$, by the definition of $\sigma \models \exists\, x : G$. Then $\tau\ (\simeq_{At(F)} \circ \approx_x)\ \rho$, and by proposition 4.3 it follows that $\tau\ (\approx_x \circ \simeq_{At(F)\cup\{x\}})\ \rho$. Now, $At(G) \subseteq At(F) \cup \{x\}$, which implies $\simeq_{At(F)\cup\{x\}} \subseteq \simeq_{At(G)}$, and therefore $\tau\ (\approx_x \circ \simeq_{At(G)})\ \rho$. Hence, there exists some behavior π such that $\tau \approx_x \pi$ and $\pi \simeq_{At(G)} \rho$. By induction hypothesis it follows that $\pi \models G$, and thus $\tau \models F$ as required. Q.E.D.

4.3 A comparison of TLA and TLA*

The propositional version of Lamport's original TLA [62] is just the fragment of pTLA* where pre-formulas are restricted to Boolean combinations of formulas v and v' for atomic propositions $v \in \mathcal{V}$. Formally, pTLA is defined by replacing clause 4 of definition 2.1 by

4'. **false** *is a pre-formula. For $v \in \mathcal{V}$, both v and v' are pre-formulas.*

The extension to quantified propositional TLA (qpTLA) is then defined as in section 4.2 (definitions 4.1 and 4.4). Every formula of (quantified) pTLA is thus a formula of (quantified) pTLA*, with the same interpretation in both logics. Starting from the observation that the "grain of atomicity" of any specification is an artefact of the level of abstraction and should therefore not be observable, Lamport's principal intention behind the definition TLA was to obtain a logic whose formulas are invariant under finite stuttering. In such a logic, implementation and composition of specifications can simply be represented by implication and conjunction.

We have shown in theorems 2.5 and 4.5 that even the extended logic qpTLA* is indeed invariant under stuttering. Does TLA* add any "harmful" expressivity to Lamport's original TLA, in the sense that its formulas can distinguish behaviors that should, intuitively, not be distinguishable? For example, TLA* allows to write formulas such as

$$\Box\Big[A \rightarrow \Box[B]_w\Big]_v$$

that are not easily expressed in TLA. Taking the expressivity of TLA as our point of reference, we now show that this is not the case: the quantified versions of pTLA and pTLA* are equi-expressive. On the other hand, TLA without quantification over flexible variables cannot express hiding and is generally considered too weak a logic for use as a specification language. We state and prove the theorem for the propositional versions of TLA and TLA*. However, the result also holds (and can be shown along the same lines) for the full first-order logics studied in section 4.7 below.

Theorem 4.6 *For every* qpTLA* *formula F there is a* qpTLA *formula F^{TLA} such that* $\models F \leftrightarrow F^{\mathrm{TLA}}$.

Proof. The idea of the proof consists in replacing complex TLA* subformulas by new proposition variables that are "bound" to the formula to be replaced. Quantification ensures that the sets of free variables of F and F^{TLA} agree.

As a first step, eliminate all quantified subformulas of F by successively choosing a fresh atomic proposition u for every (innermost) subformula $\exists\, x : G$ of F. Formally, let $F_0, F_1, \ldots$ be a sequence of formulas such that $F_0 \equiv F$, and whenever F_i is of the form $H[(\exists\, x_i : G_i)/u_i]$ for some H such that $u_i \in At(H)$ and some formula $\exists\, x_i : G_i$ such that G_i contains no subformula of the form $\exists\, y : G^*$ then F_{i+1} is H. Clearly, we have

$$\models F_i \leftrightarrow (\exists\, u_i : \Box(u_i \leftrightarrow \exists\, x_i : G_i) \wedge F_{i+1})$$

Because F_{i+1} is a smaller formula than F_i the construction must terminate, resulting in a sequence $F_0, \ldots, F_n$, for some $n \geq 0$. Furthermore, we may assume that all u_i are chosen to be different, obtaining a formula that is equivalent to the original formula F and of the form (modulo simple rearrangements of quantifiers)

$$F^* \;\equiv\; \exists\, u_0, \ldots, u_{n-1} : \Big(\bigwedge_{i=0}^{n-1} \Box(u_i \leftrightarrow \exists\, x_i : G_i)\Big) \wedge F_n$$

such that no quantifiers appear in F_n or G_i. We now successively eliminate (topmost) non-atomic formulas H such that H or $\circ H$ occurs inside a subformula $\Box[C]_v$. Formally, whenever the formula A under consideration is of the form $B[H/y]$ for some formulas

B and H such that H is not a propositional combination of (possibly primed) atomic propositions, and y occurs in a subformula $\Box[C]_v$ of B, then replace A by

$$A^\dagger \;\equiv\; \exists\, y : \Box(y \leftrightarrow H) \wedge B$$

Again, the equivalence of A and $A^\dagger$ is obvious from definition 4.4. Every step of the transformation therefore preserves the equivalence of the resulting formula and the original formula F. Since every step reduces the degree of nesting of temporal formulas inside formulas $\Box[C]_v$, this transformation must again terminate. The resulting formula is the pTLA formula F^{TLA}. Q.E.D.

To illustrate the transformations applied in the proof of theorem 4.6, consider the qpTLA* formula

$$F \;\equiv\; \Box\Big[\Box v \rightarrow \bigcirc\exists\, x : \Box[w \rightarrow \bigcirc\Box x]_w\Big]_v$$

The first step produces

$$\begin{aligned} F^* \;\equiv\; \exists\, u : &\wedge \Box(u \leftrightarrow \exists\, x : \Box[w \rightarrow \bigcirc\Box x]_w) \\ &\wedge \Box[\Box v \rightarrow u']_v \end{aligned}$$

and F^{TLA} is the TLA formula

$$\begin{aligned} \exists\, u : &\wedge \Box(u \leftrightarrow \exists\, x, y : \Box(y \leftrightarrow \Box x) \wedge \Box[w \rightarrow y']_w) \\ &\wedge \exists\, z : \Box(z \leftrightarrow \Box v) \wedge \Box[z \rightarrow u']_v \end{aligned}$$

Theorem 4.6 implies that qpTLA and qpTLA* are equi-expressive. The supposed extra expressiveness of qpTLA* can be simulated in qpTLA at the cost of introducing additional quantifiers. We believe that the advantages of using TLA* rather than TLA are twofold:

- The syntax of TLA* is defined using mutually inductive definitions of formulas and pre-formulas, instead of being divided into hierarchical layers of actions and formulas. As we have argued, the extra generality of nesting temporal operators inside brackets allows abstract (requirement) specifications to expressed more directly.

- As we have seen in chapter 2, a proof system for propositional TLA* can be given along standard lines. In particular, we were able to formulate an induction axiom (ax3) for arbitrary temporal formulas where TLA would only allow state formulas; this ability was crucial to obtain completeness. No complete axiomatization of pTLA exists in the literature; we conjecture that a complete axiom system for pTLA must include rules for quantification.

4.4 Axiomatization of qpTLA*

This section presents an axiomatization of qpTLA, as an extension of the system Σ_{TLA^*} that was studied in section 2.3. The completeness of this axiomatization will be shown in section 4.6, based on the theory of stuttering-invariant ω-automata introduced in chapter 3.

The axioms and rules for quantifiers naturally fall into three distinct categories: standard introduction and elimination rules from classical first-order logic, axioms about the interplay of quantification and temporal connectives, and introduction axioms for auxiliary variables.

Standard quantifier rules. The basic introduction and elimination rules are analogous to those of classical first-order logic.

$$(\exists\text{-I}) \quad \vdash F[G/x] \rightarrow \exists\, x : F$$

$$(\exists\text{-E}) \quad \frac{\vdash F \rightarrow G}{\vdash (\exists\, x : F) \rightarrow G} \quad (x \notin At(G))$$

In these rules, F and G can be arbitrary temporal formulas. The elimination rule requires the standard proviso that x not be free in the conclusion.

Note that our version of $(\exists\text{-I})$ is stronger than the propositional version of the corresponding rule given by Lamport [62] for TLA where G would be required to be a (non-temporal) state formula. In particular, a simple consequence of $(\exists\text{-I})$ is the temporal "comprehension theorem"

$$(\exists\text{-C}) \quad \vdash \exists\, x : \Box(x \leftrightarrow F)$$

that asserts that for any formula F there exists a proposition x that can be made to behave just as F. In a proof system for TLA, $(\exists\text{-C})$ should be added as an axiom because it can not be derived from the restricted version of $(\exists\text{-I})$ discussed above.

All quantifier theorems familiar from first-order logic can be derived from $(\exists\text{-I})$ and $(\exists\text{-E})$. For example, we have the dual introduction and elimination rules for universal quantification:

$$(\forall\text{-I}) \quad \frac{\vdash F \rightarrow G}{\vdash F \rightarrow \forall\, x : G} \quad (x \notin At(F))$$

$$(\forall\text{-E}) \quad \vdash (\forall\, x : F) \rightarrow F[G/x]$$

We will also use the analogous introduction and elimination rules for rigid quantification, which are of course just theorems of propositional logic.

Barcan-style axioms. The second group of axioms concerns the interaction of quantifiers and temporal connectives.

$(\exists\text{-}\Diamond)$	$\vdash (\Diamond \exists\, x : F) \to \exists\, x : \Diamond F$	
$(\exists\text{-}\Diamond\langle\rangle)$	$\vdash \Diamond\langle \exists\, x : F\rangle_v \to \exists\, x : \Diamond\langle F\rangle_v$	
$(\exists\text{-}\Diamond\langle\circ\rangle)$	$\vdash \Diamond\langle A \wedge \circ\exists\, x : F\rangle_v \to \exists\, x : \Diamond\langle A \wedge \circ F\rangle_v$	provided $x \notin At(A)$

The shape of these axioms is familiar from first-order modal logics (with quantification over rigid variables) where they are called "Barcan formulas" and express that the underlying first-order interpretations are the same in every world. In the case of qpTLA*, the intuition is different because quantification does not relate values at different worlds, but concerns ω-sequences of values. The original Barcan formulas are trivially valid in qpTLA*, as can be seen from the definition of rigid quantification.

Axiom $(\exists\text{-}\Diamond\langle\circ\rangle)$ asserts that quantifiers can be simultaneously moved across $\circ$ and brackets; we cannot reduce this to an application of $(\exists\text{-}\Diamond\langle\rangle)$ and an axiom of the form

$$\circ\exists\, x : F \to \exists\, x : \circ F$$

because $\exists\, x : \circ F$ is not even a pre-formula of qpTLA*, for reasons discussed following definition 4.4. Let us note that the converse implications of all axioms stated above are derivable; as an example we show the case of $(\exists\text{-}\Diamond\langle\circ\rangle)$, abbreviating subderivations that do not require quantifier reasoning by TLA*.

(1)	$\vdash F \to \exists\, x : F$	$(\exists$-I)
(2)	$\mid\!\sim \circ F \to \circ\exists\, x : F$	(1)(TLA*)
(3)	$\vdash \Diamond\langle A \wedge \circ F\rangle_v \to \Diamond\langle A \wedge \circ\exists\, x : F\rangle_v$	(2)(TLA*)
(4)	$\vdash \exists\, x : \Diamond\langle A \wedge \circ F\rangle_v \to \Diamond\langle A \wedge \circ\exists\, x : F\rangle_v$	(3)($\exists$-E)

We will therefore use the same names to denote versions of the above axioms where implication has been replaced by equivalence. Of course, the dual axioms for universal-type connectives are derivable:

$(\forall\text{-}\Box)$	$\vdash (\Box \forall\, x : F) \leftrightarrow \forall\, x : \Box F$
$(\forall\text{-}\Box[])$	$\vdash \Box[\forall\, x : F]_v \leftrightarrow \forall\, x : \Box[F]_v$
$(\forall\text{-}\Box[\circ])$	$\vdash \Box[A \to \forall\, x : \circ F]_v \leftrightarrow \forall\, x : \Box[A \to \circ F]_v$

Reconsidering the proof of theorem 4.6, which showed that every formula F of qpTLA* can be expressed by a formula F^{TLA} of qpTLA, it is easy to see that the axioms and rules introduced so far are sufficient to derive the equivalence of F and F^{TLA}. We therefore note the following corollary to theorem 4.6.

Corollary 4.7 *For every formula F of* qpTLA* *there is a formula F^{TLA} of QPTLA such that* $\vdash F \leftrightarrow F^{\mathrm{TLA}}$.

Auxiliary variables. We now present the last group of axioms, which are less familiar. We use three kinds of auxiliary variables: history and prophecy variables admit "forward" and "backward" inductive definitions to be formalized in qpTLA*. Stuttering variables are used to "stretch" the observable behavior and are the formal counterpart to using the relation $\approx_x$ instead of $=_x$ in definition 4.4. In the following axioms, I is a formula such that x does not occur in the scope of a temporal connective, N is a pre-formula such that x does not occur in the scope of a temporal connective except for $\circ$, and V is a finite set of atomic propositions such that $x \notin V$ and $At(I) \cup At(N) \subseteq V \cup \{x\}$. We assume N to be written such that $\circ$ is only applied to propositions v (written v') and, suggestively if formally somewhat sloppily, we write $\exists x' : N$ to denote rigid quantification over all primed occurrences of x in N and I' to denote the formula obtained from $\circ I$ by distributing $\circ$ over all non-temporal connectives. (Using similar reasoning as in corollary 4.7 it can be shown that I and N could be restricted to be TLA state and transition formulas, respectively.)

(**∃**-H) $(\exists x : I) \wedge \Box[\forall x : \exists x' : N]_V \rightarrow \boldsymbol{\exists}\, x : I \wedge \Box[N \wedge V' \neq V]_{V,x}$

(**∃**-P) $(\Box \exists x : I) \wedge \Box[\forall x' : I' \rightarrow \exists x : I \wedge N]_V \rightarrow \boldsymbol{\exists}\, x : \Box I \wedge \Box[N \wedge V' \neq V]_{V,x}$

(**∃**-S) $\boldsymbol{\exists}\, x : \neg x \wedge \Box[(\neg x \wedge x' \wedge V' = V) \vee (x \wedge \neg x')]_{V,x} \wedge \Box\Diamond\langle \mathbf{true} \rangle_x$

Axiom (**∃**-H) asserts that there exists some x that initially satisfies I and that changes only if V changes, satisfying N, if I can be satisfied initially and N can be satisfied whenever V changes. In particular, if N defines a unique value of x' for any given V, V', and x, then (**∃**-H) corresponds to an inductive definition of x. Axiom (**∃**-P) is almost dual: if always there is some x satisfying I, and whenever V changes then for every choice of x' satisfying I' there is some x that satisfies I and N, then there exists some x that always satisfies I, satisfies N whenever V or x change value, and that changes only if V does. Axiom (**∃**-P) is not the exact dual of (**∃**-H) because behaviors have an initial but no final state. Finally, axiom (**∃**-S) introduces a stuttering variable x that changes infinitely often, and at least once between any change of V (because x must be true whenever V changes and is reset to false at every such transition).

This completes the presentation of the quantifier axioms and rules. It remains to show that the system obtained in this way is sound.

Theorem 4.8 *The proof system obtained from* Σ_{TLA^*} *by adding* (**∃**-*I*), (**∃**-*E*), (**∃**-$\Diamond$), (**∃**-$\Diamond\langle\rangle$), (**∃**-$\Diamond\langle\circ\rangle$), (**∃**-*H*), (**∃**-*P*), *and* (**∃**-*S*) *is sound for* qpTLA*.

Proof. Extending theorem 2.6, we need only consider the newly introduced axioms and rules.

($\exists$-I). Assume that $\sigma = s_0 s_1 \dots$ is some behavior such that $\sigma \models F[G/x]$ and define the behavior $\tau = t_0 t_1 \dots$ by

$$t_i(x) \triangleq \left\{ \begin{array}{ll} \mathrm{tt} & \text{if } \sigma[i..] \models G \\ \mathrm{ff} & \text{otherwise} \end{array} \right\} \quad \text{and} \quad t_i(y) \triangleq s_i(y) \text{ for all } y \not\equiv x$$

A straightforward induction on the shape of formula F shows that $\tau \models F$. Now, clearly we have $\tau =_x \sigma$, implying that $\tau \approx_x \sigma$, and therefore $\sigma \models \exists x : F$.

($\exists$-E). Assume that $\vdash F \to G$ where $x \notin At(G)$, and let σ be any behavior such that $\sigma \models \exists x : F$. Choose some $\tau \approx_x \sigma$ such that $\tau \models F$; such τ exists by definition 4.4. By induction hypothesis, we may assume $\models F \to G$, and therefore $\tau \models G$. Now, since $\tau \approx_x \sigma$, there exist behaviors π and ρ such that $\tau \simeq \pi =_x \rho \simeq \sigma$. From $\tau \models G$, $\tau \simeq \pi$, and theorem 4.5 we obtain $\pi \models G$. From $\pi =_x \rho$ and $x \notin At(G)$ it follows easily that $\rho \models G$. Finally, using $\rho \simeq \sigma$ and theorem 4.5, we have $\sigma \models G$ as desired.

($\exists$-$\Diamond$). Assume that σ is a behavior such that $\sigma \models \Diamond \exists x : F$, say $\sigma[i..] \models \exists x : F$. Choose some behavior $\tau \approx_x \sigma[i..]$ such that $\tau \models F$ and let ρ denote the behavior $\sigma[..i] \circ \tau$. Observing that $\rho \approx_x \sigma$ and that $\rho[i..] \models F$, it follows that $\sigma \models \exists x : \Diamond F$.

($\exists$-$\Diamond\langle\rangle$). The proof is similar to the case of ($\exists$-$\Diamond$) above.

($\exists$-$\Diamond\langle\circ\rangle$). Assume that $\sigma = s_0 s_1 \dots$ is a behavior such that $\sigma \models \Diamond\langle A \wedge \circ\exists x : F\rangle_v$, say $\sigma[i..] \mathrel{|\!\approx} v' \neq v \wedge A \wedge \circ\exists x : F$. In particular, $\sigma[i+1..] \models \exists x : F$, so choose some behavior $\tau \approx_x \sigma[i+1..]$ such that $\tau \models F$. From the assumption that $x \notin At(A)$ and from the choice of τ it follows that $\sigma[i..] \simeq_{At(A)} s_i\tau$ and $\sigma[i+1..] \simeq_{At(A)} \tau$. Therefore, theorem 2.5 implies that $s_i\tau \mathrel{|\!\approx} v' \neq v \wedge A \wedge \circ F$. Define the behavior $\rho \triangleq \sigma[..i]\tau$. Observing that $\rho[i..] = s_i\tau$, it follows that $\rho \models \Diamond\langle A \wedge \circ F\rangle_v$. Moreover, we have $\rho \approx_x \sigma$, and therefore $\sigma \models \exists x : \Diamond\langle A \wedge \circ F\rangle_v$ as required.

($\exists$-H). Assume that $\sigma = s_0 s_1 \dots$ is some behavior such that

$$\sigma \models (\exists x : I) \wedge \Box[\forall x : \exists x' : N]_V$$

where I, N, and V are as in the statement of axiom ($\exists$-H). It is enough to define a behavior $\tau = t_0 t_1 \dots$ such that $\tau =_x \sigma$ and

$$\tau \models I \wedge \Box[N \wedge V' \neq V]_{V,x} \tag{4.1}$$

Because $\tau =_x \sigma$, we only need to define $t_i(x)$, inductively for all $i \in \mathbb{N}$. We define

$$t_0(x) \triangleq \begin{cases} \mathrm{tt} & \text{if } \sigma \models I[\mathbf{true}/x] \\ \mathrm{ff} & \text{otherwise} \end{cases}$$

$$t_{i+1}(x) \triangleq \begin{cases} t_i(x) & \text{if } s_i(v) = s_{i+1}(v) \text{ for all } v \in V \\ \mathrm{tt} & \text{else if } \sigma[i..]^* \mathrel{|\!\approx} N[\mathbf{true}/x'] \\ \mathrm{ff} & \text{otherwise} \end{cases}$$

where $\sigma[i..]^*$ denotes the behavior $t_i s_{i+1} s_{i+2} \ldots$. Clearly, this definition ensures that the value of x changes only if some variable in V changes value. Moreover, the syntactic conditions on I and N ensure that (4.1) also holds because only the value $t_0(x)$ is required to determine whether $\tau \models I$ and, similarly, only $t_i(x)$ and $t_{i+1}(x)$ are necessary to determine whether $\tau[i..] \approx N$, and all of these values were chosen appropriately.

(**∃**-P). Assume that $\sigma = s_0 s_1 \ldots$ is a behavior such that, for all $i \in \mathbb{N}$

1. $\sigma[i..] \models \exists x : I$ and
2. $\sigma[i..] \approx \forall x' : I' \rightarrow \exists x : I \wedge N$ whenever $s_{i+1}(v) \neq s_i(v)$ for some $v \in V$.

Writing s^{tt} (resp., s^{ff}) for the states that agree with s except that x is assigned tt (resp., ff), we construct a dag (directed acyclic graph) of pairs (i, b) where $i \in \mathbb{N}$ and $b \in \{\mathsf{tt}, \mathsf{ff}\}$ as follows: the node (i, b) appears in the dag if $s_i^b s_{i+1} s_{i+2} \ldots \models I$. Node (j, c) is a successor of node (i, b) iff $j = i + 1$ and either $b = c$ and $s_{i+1}(v) = s_i(v)$ holds for all $v \in V$, or $s_i^b s_{i+1}^c \ldots \approx N$.

Condition (1) above ensures that the dag contains some node (i, b) for every $i \in \mathbb{N}$, it therefore has infinitely many nodes. On the other hand, condition (2) guarantees that every node $(i+1, c)$ has a predecessor node (i, b). It follows that the dag contains finite paths of arbitrary length; it is (obviously) also of finite width. König's lemma thus ensures the existence of an infinite path $\pi = (0, b_0)(1, b_1) \ldots$ in the dag. Choose some such path and define the behavior $\tau = t_0 t_1 \ldots$ by setting $t_i(x) \triangleq b_i$ and $t_i(y) = s_i(y)$ for all $y \not\equiv x$. Obviously, this definition ensures that $\tau =_x \sigma$ and that $t_i(x) \neq t_{i+1}(x)$ only if $t_i(v) \neq t_{i+1}(v)$ for some $v \in V$. Finally, the definition of the dag and the syntactic conditions on I and N ensure that

$$\tau \models \Box I \wedge \Box[N \wedge V' \neq V]_{V,x}$$

which implies the assertion.

(**∃**-S). Assume that $\sigma = s_0 s_1 \ldots$ is some behavior, and define the behavior $\tau = t_0 t_1 \ldots$ by

$$\begin{array}{lcllcll} t_{2i}(x) & \triangleq & \mathsf{ff} & t_{2i+1}(x) & \triangleq & \mathsf{tt} & \\ t_{2i}(y) & \triangleq & s_i(y) & t_{2i+1}(y) & \triangleq & s_i(y) & \text{(for } y \not\equiv x\text{)} \end{array}$$

This definition ensures that $\tau \approx_x \sigma$ and that $t_{2i+1}(v) = t_{2i}(v)$ for all $v \in V$ and all $i \in \mathbb{N}$. Clearly, $t_i(X) \neq t_{i+1}(x)$ holds for infinitely many $i \in \mathbb{N}$. Altogether, this implies

$$\tau \models \neg x \wedge \Box[(\neg x \wedge x' \wedge V' = V) \vee (x \wedge \neg x')]_{V,x} \wedge \Box\Diamond\langle \mathbf{true} \rangle_x$$

and the assertion follows. Q.E.D.

4.5 Encoding Edge Automata In qpTLA*

We now relate the theory of stuttering-invariant ω-regular languages presented in chapter 3 and qpTLA* formulas. We first describe an encoding of ω automata by qpTLA* formulas such that the automaton accepts (up to stuttering equivalence) precisely the models of the encoding formula. Next, we show that the correctness of the principal automata-theoretic constructions considered in chapter 3 can be reestablished by proving the corresponding equivalences in the proof system of section 4.4.

For a given edge automaton $\mathcal{A} = (Q, q_0, \delta, C)$ we now define a *characteristic formula* $\chi_{\mathcal{A}}$ of propositional TLA that (essentially) defines $\mathcal{L}(\mathcal{A})$. Assume that $Q = \{q_0, \dots, q_n\}$ is the set of locations of automaton $\mathcal{A}$ and that V is the set of atomic propositions that occur in the transition and acceptance conditions of $\mathcal{A}$. To encode the current location of $\mathcal{A}$, we use $m+1$ atomic formulas, denoted by $\overline{q}_0, \dots, \overline{q}_n \notin V$, and write

$$Q_i \;\equiv\; \overline{q}_i \wedge \bigwedge_{j \neq i} \neg \overline{q}_i \qquad (i = 0, \dots, n)$$

to express that the current location of $\mathcal{A}$ is q_i. By $\overline{q}$ we denote the list $\overline{q}_0, \dots, \overline{q}_n$.

The characteristic formula $\chi_{\mathcal{A}}$ of a Büchi automaton $\mathcal{A}$ is defined as follows:

$$\begin{aligned}
\chi_{\mathcal{A}} &\equiv \exists \overline{q} : \varphi_{\mathcal{A}} \\
\varphi_{\mathcal{A}} &\equiv \mathit{init}_{\mathcal{A}} \wedge \Box[\mathit{trans}_{\mathcal{A}}]_{\overline{q}, V} \wedge \mathit{acc}_{\mathcal{A}} \\
\mathit{init}_{\mathcal{A}} &\equiv \bigvee_{i=0}^{n} Q_i \wedge \delta(q_0, q_i) \\
\mathit{trans}_{\mathcal{A}} &\equiv \bigvee_{i,j=0}^{n} Q_i \wedge Q_j' \wedge \delta(q_i, q_j)' \\
\mathit{acc}_{\mathcal{A}} &\equiv \vee \bigvee_{k,l=0}^{n} \Box\Diamond\langle Q_k \wedge Q_l' \wedge C(q_k, q_l)' \rangle_{\overline{q}, V} \\
& \vee \bigvee_{k=0}^{n} \Box\Diamond(Q_k \wedge \delta(q_k, q_k) \wedge C(q_k, q_k))
\end{aligned}$$

The characteristic formula $\chi_{\mathcal{A}}$ for a Rabin automaton $\mathcal{A}$ with acceptance condition $C = \{(R_1, G_1), \dots, (R_m, G_m)\}$ is defined as for a Büchi automaton, except that the last conjunct of $\varphi_{\mathcal{A}}$ becomes

$$\begin{aligned}
\mathit{acc}_{\mathcal{A}} \equiv \bigvee_{j=1}^{m} &\wedge \bigwedge_{k,l=0}^{n} \Diamond\Box[\neg(Q_k \wedge Q_l' \wedge R_j(q_k, q_l)')]_{\overline{q}, V} \\
&\wedge \bigvee_{k,l=0}^{n} \Box\Diamond\langle Q_k \wedge Q_l' \wedge G_j(q_k, q_l)' \rangle_{\overline{q}, V}
\end{aligned}$$

Finally, if $\mathcal{A}$ is a Streett automaton whose acceptance condition is given as $C = \{(R_1, G_1), \ldots, (R_m, G_m)\}$, the characteristic formula is obtained by setting

$$\begin{array}{rcl} acc_{\mathcal{A}} & \equiv & \wedge \bigwedge_{j=1}^{m} \vee \bigwedge_{k,l=0}^{n} \Diamond\Box[\neg(Q_k \wedge Q_l' \wedge R_j(q_k, q_l)')]_{\overline{q},V} \\ & & \qquad\quad \vee \bigvee_{k,l=0}^{n} \Box\Diamond\langle Q_k \wedge Q_l' \wedge G_j(q_k, q_l)'\rangle_{\overline{q},V} \\ & & \wedge\ \Box\Diamond\langle \mathbf{true}\rangle_{\overline{q},V} \end{array}$$

For any edge automaton $\mathcal{A}$, standard invariant reasoning shows that

$$\models \varphi_{\mathcal{A}} \rightarrow \Box loc_{\mathcal{A}}$$

where the formula $loc_{\mathcal{A}}$ is defined as

$$loc_{\mathcal{A}} \quad \equiv \quad \bigvee_{i=0}^{n} Q_i$$

and expresses that at every instant, precisely one location of $\mathcal{A}$ is active.

For a Büchi automaton $\mathcal{A}$, we now show that its characteristic formula $\chi_{\mathcal{A}}$ defines the V-stuttering closure of $\mathcal{L}(\mathcal{A})$. In particular, if $\mathcal{A}$ is an SI_V automaton (and therefore $\mathcal{L}(\mathcal{A})$ is closed w.r.t. V-stuttering) then $\chi_{\mathcal{A}}$ is true of precisely the behaviors in $\mathcal{L}(\mathcal{A})$.

Theorem 4.9 *Assume that $\mathcal{A}$ is a Büchi automaton and that σ is some behavior. Then $\sigma \models \chi_{\mathcal{A}}$ holds iff $\sigma \in \mathcal{SC}_V(\mathcal{L}(\mathcal{A}))$.*

Proof. "$\Rightarrow$": Assume that $\sigma \models \chi_{\mathcal{A}}$. Choose some $\sigma^* \approx_{\overline{q}} \sigma$ such that $\sigma^* \models \varphi_{\mathcal{A}}$, and let $\sigma^\natural$ be some $(\overline{q}, V)$-stuttering free variant of σ^*. Because $\varphi_{\mathcal{A}}$, like all TLA formulas, is invariant under stuttering (theorem 2.5), it follows that $\sigma^\natural \models \varphi_{\mathcal{A}}$. Moreover, we have $\sigma \simeq_V \sigma^*$ as well as $\sigma^* \simeq_{\overline{q},V} \sigma^\natural$, and since $\simeq_V$ is a coarser equivalence relation than $\simeq_{\overline{q},V}$, it follows that $\sigma \simeq_V \sigma^\natural$. Now define the behavior $\tau = t_0 t_1 \ldots$ by duplicating all states of $\sigma^\natural$ that satisfy

$$Q_k \wedge \delta(q_k, q_k) \wedge C(q_k, q_k)$$

for some $k \in \{0, \ldots, n\}$. Clearly, this ensures $\sigma^\natural \simeq_V \tau$, hence also $\tau \models \varphi_{\mathcal{A}}$ and $\sigma \simeq_V \tau$. We will now show that $\tau \in \mathcal{L}(\mathcal{A})$; this implies $\sigma \in \mathcal{SC}_V(\mathcal{L}(\mathcal{A}))$ as required.

Define the sequence $\rho = r_0 r_1 \ldots$ of $\mathcal{A}$-locations by $r_0 = q_0$ and $r_{i+1} = q_k$ iff $t_i \models Q_k$ (observe that r_{i+1} is well-defined because $\tau \models \Box loc_{\mathcal{A}}$, and because $t_i \models Q_k$ can hold for at most one k). We first prove that ρ is a run of $\mathcal{A}$ over τ.

1. By definition, we have $r_0 = q_0$.

2. Let $r_1 = q_k$. Because $\tau \models init_{\mathcal{A}}$, it follows that $t_0 \models \delta(q_0, q_k)$, hence $r_0 \xrightarrow{t_0} r_1$.

3. We now prove that $r_{i+1} \xrightarrow{t_{i+1}} r_{i+2}$ holds for all $i \geq 0$. Let $r_{i+1} = q_k$ and $r_{i+2} = q_l$. We distinguish the following cases:

   (a) If $t_i \neq_{\overline{q},V} t_{i+1}$ it follows that $(t_i, t_{i+1}) \models trans_{\mathcal{A}}$ and therefore $q_k \xrightarrow{t_{i+1}} q_l$ as desired.

   (b) If t_{i+1} was obtained by duplicating t_i then we must have $q_k = q_l$ (by the definition of Q_k). Moreover, the definition of τ implies that in this case we have $t_i \models Q_k \wedge \delta(q_k, q_k)$, and the assertion again holds.

   (c) Otherwise t_i and t_{i+1} must be consecutive states from $\sigma^\natural$, but $t_i =_{\overline{q},V} t_{i+1}$. Since $\sigma^\natural$ was assumed to be $(\overline{q}, V)$-stuttering free, this can only be the case if $\sigma^\natural$, and therefore τ, end in infinite stuttering w.r.t. $(\overline{q}, V)$, that is, $t_j = t_i$ for all $j \geq i$. Now, $\tau \models acc_{\mathcal{A}}$, and therefore we must have

   $$\tau \models \bigvee_{k=0}^{n} \Box\Diamond(Q_k \wedge \delta(q_k, q_k) \wedge C(q_k, q_k))$$

   The definition of Q_k and the assumption that $t_j = t_i$ for all $j \geq i$ therefore imply $q_k = q_l$ and $t_{i+1} \models \delta(q_k, q_k)$, and thus the assertion.

It remains to prove that the run ρ over τ is accepting, that is, $t_i \models C(r_i, r_{i+1})$ for infinitely many $i \in \mathbb{N}$. Now, we have $\tau \models acc_{\mathcal{A}}$, and there are again two cases: if

$$\tau \models \bigvee_{k,l=0}^{n} \Box\Diamond\langle Q_k \wedge Q_l' \wedge C(q_k, q_l)'\rangle_{\overline{q},V}$$

then the assertion follows immediately from the definition of the locations r_i. On the other hand, if

$$\tau \models \bigvee_{k=0}^{n} \Box\Diamond(Q_k \wedge \delta(q_k, q_k) \wedge C(q_k, q_k))$$

then every state satisfying $Q_k \wedge \delta(q_k, q_k) \wedge C(q_k, q_k)$ has been duplicated in τ, and therefore there are infinitely many $i \in \mathbb{N}$ such that $r_{i+1} = r_i$ and $t_i \models C(r_i, r_i)$. This suffices.

"$\Leftarrow$": Because $\chi_{\mathcal{A}}$, like every PTLA formula, is invariant under V-stuttering by theorem 4.5, it is enough to prove that $\sigma \models \chi_{\mathcal{A}}$ whenever $\sigma \in \mathcal{L}(\mathcal{A})$. So let $\rho = r_0 r_1 \ldots$ be an accepting run of $\mathcal{A}$ over σ, and define the $\overline{q}$-variant $\tau = t_0 t_1 \ldots$ of σ by $t_i(\overline{q}_k) = \mathtt{tt}$ iff $r_{i+1} = q_k$. From the definition of $\varphi_{\mathcal{A}}$, it is straightforward to prove that $\tau \models \varphi_{\mathcal{A}}$, and therefore $\sigma \models \chi_{\mathcal{A}}$. In particular, $\tau \models acc_{\mathcal{A}}$ even if ρ eventually remains in some location q_k, because then τ must satisfy $\Box\Diamond(Q_k \wedge \delta(q_k, q_k) \wedge C(q_k, q_k))$. Q.E.D.

For the other types of edge automata we have similar results, with a slight technical restriction: in the proof of theorem 4.9, it was essential to duplicate states that satisfied an acceptance formula at a loop, in order to ensure the existence of some stuttering variant of the given behavior that is accepted by the Büchi automaton. For Rabin and Streett automata, this would be more difficult, because duplication of states that satisfied some condition at a "green" loop might introduce additional "red" loops. We therefore consider only loop-free Rabin and Streett automata; as noted in section 3.2, for every edge automaton there is an equivalent loop-free edge automaton of the same type.

Theorem 4.10 *Assume that $\mathcal{A}$ is a loop-free Rabin or Streett automaton and that σ is some behavior. Then $\sigma \models \chi_{\mathcal{A}}$ holds iff $\sigma \in \mathcal{SC}_V(\mathcal{L}(\mathcal{A}))$.*

Proof. The proof follows the same line of reasoning as the proof of theorem 4.9, except that $\sigma^\natural$ is guaranteed (by the definition of $acc_{\mathcal{A}}$) to contain no stuttering transitions w.r.t. $(\overline{q}, V)$, and no duplication of states is necessary for the definition of τ. Q.E.D.

Theorems 4.9 and 4.10 establish close correspondences between the language of edge automata and the models of its characteristic formulas. We will now reconsider some of the constructions introduced in the preceding sections from a deductive point of view.

Theorem 4.11 *Assume that $\mathcal{A}$ is a loop-free Streett automaton and that $\mathcal{B}$ is the Büchi automaton constructed in the proof of theorem 3.2. Then $\vdash \chi_{\mathcal{A}} \leftrightarrow \chi_{\mathcal{B}}$.*

Proof. "$\Rightarrow$": Assume that $\chi_{\mathcal{A}} \equiv \exists \overline{q}_{\mathcal{A}} : \varphi_{\mathcal{A}}$ and that $\chi_{\mathcal{B}} \equiv \exists \overline{q}_{\mathcal{B}} : \varphi_{\mathcal{B}}$. For the proof of $\vdash \chi_{\mathcal{A}} \rightarrow \chi_{\mathcal{B}}$ it suffices, by rule ($\exists$-E), to show $\vdash \varphi_{\mathcal{A}} \rightarrow \exists \overline{q}_{\mathcal{B}} : \varphi_{\mathcal{B}}$. By the definition of $\varphi_{\mathcal{A}}$, the temporal logic equivalence mentioned in the proof of theorem 3.2, and the completeness of unquantified propositional TLA*, it follows that

$$\vdash \varphi_{\mathcal{A}} \rightarrow \Diamond\Box Z \tag{4.2}$$

where

$$Z \equiv \bigwedge_{j=1}^{m} \left(\Diamond\langle R_j \rangle_{\overline{q}_{\mathcal{A}}, V} \rightarrow \Diamond\langle G_j \rangle_{\overline{q}_{\mathcal{A}}, V} \right)$$

Now, given the current location of $\mathcal{A}$, define the location of $\mathcal{B}$ as a history variable, as follows:

- The initial condition HI requires $\overline{q}_{\mathcal{B}}$ to encode the location $(\overline{q}_{\mathcal{A}}, \bot, \bot)$.
- In every transition, the next-state relation HN sets the first component of $\overline{q}'_{\mathcal{B}}$ to the location encoded by $\overline{q}_{\mathcal{A}}$ and sets the R- and G-components of $\overline{q}'_{\mathcal{B}}$ to $\bot$ as long as $\Box Z$ is false, and otherwise updates them as indicated in the proof of theorem 3.2.

The semantic argument given there establishes the validity of

$$\varphi_{\mathcal{A}} \wedge \Diamond\Box Z \wedge HI \wedge \Box[HN \wedge (\overline{q}_{\mathcal{A}}, V)' \neq (\overline{q}_{\mathcal{A}}, V)]_{\overline{q}_{\mathcal{B}},\overline{q}_{\mathcal{A}},V} \rightarrow \varphi_{\mathcal{B}} \tag{4.3}$$

hence this implication is also provable, by the completeness proof of section 2.4. Standard quantifier reasoning shows

$$\begin{aligned} &\vdash \varphi_{\mathcal{A}} \wedge \Diamond\Box Z \wedge \big(\exists \overline{q}_{\mathcal{B}} : HI \wedge \Box[HN \wedge (\overline{q}_{\mathcal{A}}, V)' \neq (\overline{q}_{\mathcal{A}}, V)]_{\overline{q}_{\mathcal{B}},\overline{q}_{\mathcal{A}},V}\big) \\ &\quad \rightarrow \exists \overline{q}_{\mathcal{B}} : \chi_{\mathcal{B}} \end{aligned} \tag{4.4}$$

and the desired result,

$$\vdash \varphi_{\mathcal{A}} \rightarrow \exists \overline{q}_{\mathcal{B}} : \varphi_{\mathcal{B}} \tag{4.5}$$

follows by an application of ($\exists$-H) and (4.2).

"$\Leftarrow$": For the other direction, it suffices to define $\overline{q}_{\mathcal{A}}$ by a refinement mapping that projects $\overline{q}_{\mathcal{B}}$ to its first component and follow the semantic proof given for theorem 3.2. In particular, note that the assumption that $\mathcal{A}$ is loop-free implies that so is the resulting Büchi automaton $\mathcal{B}$. Therefore, the disjunct

$$\bigvee_{k=0}^{n} \Box\Diamond(Q_k \wedge \delta_{\mathcal{B}}(q_k, q_k) \wedge C(q_k, q_k))$$

of subformula $acc_{\mathcal{B}}$ cannot hold of any behavior satisfying $\varphi_{\mathcal{B}}$, and the subformula

$$\Box\Diamond\langle \mathbf{true} \rangle_{\overline{q}_{\mathcal{A}},V}$$

must in fact be implied by $\varphi_{\mathcal{B}}$ under the given refinement mapping. Q.E.D.

We now turn to the determinization construction of section 3.3 and show that the characteristic formulas of the respective automata are provably equivalent. This result will be the key to proving the completeness of our quantifier rules.

Theorem 4.12 *Let $\mathcal{A}$ be some Büchi edge automaton, and $\mathcal{D}$ be the corresponding loop-free and deterministic Rabin edge automaton according to the construction of figure 3.3. Then $\vdash \chi_{\mathcal{A}} \leftrightarrow \chi_{\mathcal{D}}$.*

Proof. Assume that $\mathcal{A} = (Q^{\mathcal{A}}, q_0^{\mathcal{A}}, \delta^{\mathcal{A}}, C^{\mathcal{A}})$ and $\mathcal{D} = (Q^{\mathcal{D}}, q_0^{\mathcal{D}}, \delta^{\mathcal{D}}, C^{\mathcal{D}})$ are the Büchi and Rabin automata where $Q^{\mathcal{A}} = \{q_0^{\mathcal{A}}, \ldots, q_n^{\mathcal{A}}\}$ and $Q^{\mathcal{D}} = \{q_0^{\mathcal{D}}, \ldots, q_d^{\mathcal{A}}\}$ are the respective sets of locations, and where $C^{\mathcal{D}} = \{(R_1, G_1), \ldots, (R_n, G_n)\}$ is the acceptance condition of $\mathcal{D}$. Also assume that $\chi_{\mathcal{A}} \equiv \exists \overline{q}_{\mathcal{A}} : \varphi_{\mathcal{A}}$ and that $\chi_{\mathcal{D}} \equiv \exists \overline{q}_{\mathcal{D}} : \varphi_{\mathcal{D}}$. We write $Q_{\mathcal{A},i} \equiv Q_i[\overline{q}_{\mathcal{A}}/\overline{q}]$ (for $i = 1, \ldots, n$) and $Q_{\mathcal{D},j} \equiv Q_j[\overline{q}_{\mathcal{D}}/\overline{q}]$ (for $j = 1, \ldots, d$) for the state predicates that encode the current locations of $\mathcal{A}$ and $\mathcal{D}$, respectively.

Similar to the proof of theorem 3.6, if q_i is a location of $\mathcal{D}$ and $k \in \{1, \ldots, n\}$ we write $q_i.k$ to denote the node numbered k in the n-tree of location q_i, provided such a node exists. We also write $\lambda(q_i.k)$ to denote the label set of node $q_i.k$ and $child(q_i.k)$ to denote the set of children nodes of node $q_i.k$, with the convention that $\lambda(q_i.k) = child(q_i.k) = \emptyset$ if q_i does not contain a node numbered k. For $k = 1, \ldots, n$, we introduce the state predicates

$$\begin{array}{rcl}\Lambda_k & \equiv & \bigvee_{i=1}^{d} \left(Q_{\mathcal{D},i} \wedge \bigvee_{j \in \lambda(q_i.k)} Q_{\mathcal{A},j} \right) \\ \Lambda_k^c & \equiv & \bigvee_{i=1}^{d} \left(Q_{\mathcal{D},i} \wedge \bigvee_{l \in child(q_i.k)} \bigvee_{j \in \lambda(q_i.l)} Q_{\mathcal{A},j} \right)\end{array}$$

Intuitively, Λ_k holds if $\overline{q}_{\mathcal{A}}$ encodes some $\mathcal{A}$-location contained in the label set of the node numbered k in the $\mathcal{D}$-location encoded by $\overline{q}_{\mathcal{D}}$. Similarly, Λ_k^c holds if $\overline{q}_{\mathcal{A}}$ encodes some $\mathcal{A}$-location contained in the label set of some child of the node numbered k in the $\mathcal{D}$-location encoded by $\overline{q}_{\mathcal{D}}$.

"$\Rightarrow$": As a first step, we introduce a Boolean stuttering variable s (different from the variables in V, $\overline{q}_{\mathcal{A}}$, and $\overline{q}_{\mathcal{D}}$) that ticks infinitely often: ($\exists$-S) and simple temporal logic prove

$$\vdash \exists s : \Box\Diamond\langle \mathbf{true} \rangle_s \tag{4.6}$$

We use a history variable to compute the current location of a run of $\mathcal{D}$ over some stuttering variant of the underlying behavior. Formally, we claim that ($\exists$-H) justifies

$$\vdash init_{\mathcal{A}} \wedge \Box[trans_{\mathcal{A}}]_{\overline{q}_{\mathcal{A}},V} \rightarrow \exists \overline{q}_{\mathcal{D}} : init_{\mathcal{D}} \wedge \Box[HN]_{\overline{q}_{\mathcal{D}},\overline{q}_{\mathcal{A}},V,s} \tag{4.7}$$

where the transition formula HN is defined as

$$\begin{array}{rcl}HN & \equiv & \vee\ (trans_{\mathcal{A}} \vee acc\_loop_{\mathcal{A}}) \wedge trans_{\mathcal{D}} \\ & & \vee\ \neg(trans_{\mathcal{A}} \vee acc\_loop_{\mathcal{A}}) \wedge \overline{q}'_{\mathcal{D}} = \overline{q}_{\mathcal{D}} \\ acc\_loop_{\mathcal{A}} & \equiv & (\overline{q}_{\mathcal{A}}, V)' = (\overline{q}_{\mathcal{A}}, V) \wedge \bigvee_{i=1}^{n} \left(Q_{\mathcal{A},i} \wedge \delta^{\mathcal{A}}(q_i, q_i) \wedge C^{\mathcal{A}}(q_i, q_i) \right)\end{array}$$

This claim follows from the construction of $\mathcal{D}$, which ensures that $\delta^{\mathcal{D}}$ is a total relation. We will now prove that

$$\models \varphi_{\mathcal{A}} \wedge init_{\mathcal{D}} \wedge \Box[HN]_{\overline{q}_{\mathcal{D}},\overline{q}_{\mathcal{A}},V,s} \wedge \Box\Diamond\langle \mathbf{true} \rangle_s \rightarrow \varphi_{\mathcal{D}} \tag{4.8}$$

Assertion (4.8) is proven semantically. First, standard invariant reasoning shows that

$$\models \varphi_{\mathcal{A}} \wedge init_{\mathcal{D}} \wedge \Box[HN]_{\overline{q}_{\mathcal{D}},\overline{q}_{\mathcal{A}},V,s} \rightarrow \Box\Lambda_1$$

because for every move of $\mathcal{D}$, the label of the root of the target location contains the set of $\mathcal{A}$-locations that are reachable from the set of $\mathcal{A}$-locations contained in the label of the root of the source location. The definition of HN obviously implies

$$\models \varphi_{\mathcal{A}} \wedge init_{\mathcal{D}} \wedge \Box[HN]_{\overline{q}_{\mathcal{D}},\overline{q}_{\mathcal{A}},V,s} \rightarrow \Box[trans_{\mathcal{D}}]_{\overline{q}_{\mathcal{D}},V}$$

It therefore suffices to prove

$$\models \varphi_{\mathcal{A}} \wedge init_{\mathcal{D}} \wedge \Box[HN]_{\overline{q}_{\mathcal{D}},\overline{q}_{\mathcal{A}},V,s} \wedge \Box\Diamond\langle \mathbf{true}\rangle_s \rightarrow acc_{\mathcal{D}}$$

Because $\mathcal{D}$ is loop-free and s, if not $(\overline{q}_{\mathcal{A}}, V)$, changes value infinitely often, the sequence of values assumed by the variables $\overline{q}_{\mathcal{D}}$ encodes an infinite run of $\mathcal{D}$ (as in the proof of theorems 4.9 and 4.10) over a stuttering variant of the underlying behavior. This run must be accepting, as we have shown in the proof of theorem 3.6 by induction on the depth of the n-trees that occur in the locations of $\mathcal{D}$ during the run.

Because the implication in (4.8) is a quantifier-free formula, the completeness theorem 2.16 ensures that it is provable, and we obtain

$$\vdash \varphi_{\mathcal{A}} \wedge init_{\mathcal{D}} \wedge \Box[trans_{\mathcal{D}}]_{\overline{q}_{\mathcal{D}},\overline{q}_{\mathcal{A}},V,s} \wedge \Box\Diamond\langle \mathbf{true}\rangle_s \rightarrow \varphi_{\mathcal{D}} \tag{4.9}$$

from which $\vdash \chi_{\mathcal{A}} \rightarrow \chi_{\mathcal{D}}$ follows with the help of (4.6), (4.7), and ordinary quantifier reasoning.

"$\Leftarrow$": The implication $\chi_{\mathcal{D}} \rightarrow \chi_{\mathcal{A}}$ follows by propositional logic and ($\exists$-E) from

$$\vdash \varphi^j_{\mathcal{D}} \rightarrow \exists\, \overline{q}_{\mathcal{A}} : \varphi_{\mathcal{A}} \qquad (j = 1, \ldots, n) \quad \text{where} \tag{4.10}$$

$$\begin{aligned} \varphi^j_{\mathcal{D}} \;\equiv\; & \wedge\; init_{\mathcal{D}} \wedge \Box[trans_{\mathcal{D}}]_{\overline{q}_{\mathcal{D}},V} \\ & \wedge \bigwedge_{k,l=0}^{d} \Diamond\Box[\neg(Q_{\mathcal{D},k} \wedge Q'_{\mathcal{D},l} \wedge R_j(q_k,q_l)')]_{\overline{q}_{\mathcal{D}},V} \\ & \wedge \bigvee_{k,l=0}^{d} \Box\Diamond\langle Q_{\mathcal{D},k} \wedge Q'_{\mathcal{D},l} \wedge G_j(q_k,q_l)'\rangle_{\overline{q}_{\mathcal{D}},V} \end{aligned}$$

which we now prove. For $j = 1, \ldots, n$, let D_j denote the set

$$D_j \;=\; \{\, i \in \{0, \ldots, d\} : \text{location } q_i \text{ of } \mathcal{D} \text{ contains a node numbered } j \,\}$$

of (indexes of) locations q_i of $\mathcal{D}$ for which $q_i.j$ exists, and let Z_j denote the formula

$$Z_j \;\equiv\; \bigvee_{i \in D_j} \left(Q_{\mathcal{D},i} \wedge \bigwedge_{k,l=0}^{d} \Box[\neg(Q_{\mathcal{D},k} \wedge Q'_{\mathcal{D},l} \wedge R_j(q_k,q_l)')]_{\overline{q}_{\mathcal{D}},V} \right)$$

Formula Z_j is true if $\overline{q}_{\mathcal{D}}$ encodes a location of $\mathcal{D}$ that contains a node numbered j, and if $\mathcal{D}$ never follows a red edge (with respect to the j-th acceptance condition). We will prove below that $\varphi^j_{\mathcal{D}}$ implies that formula Z_j eventually becomes true, and then stays true forever.

We can now define the current location of an accepting run of $\mathcal{A}$ using a prophecy variable. Specifically, we claim that ($\exists$-P) shows

$$\vdash \varphi^j_{\mathcal{D}} \rightarrow \exists\, \overline{q}_{\mathcal{A}} : \Box I \wedge \Box[PN]_{\overline{q}_{\mathcal{A}},\overline{q}_{\mathcal{D}},V} \tag{4.11}$$

where I and PN are defined by

$$
\begin{array}{rcl}
I & \equiv & \Lambda_1 \wedge (Z_j \rightarrow \Lambda_j) \\
PN & \equiv & \vee\, (\exists \overline{q}_{\mathcal{A}} : PN_1) \wedge PN_1 \\
 & & \vee\, \neg(\exists \overline{q}_{\mathcal{A}} : PN_1) \wedge (\exists \overline{q}_{\mathcal{A}} : PN_2) \wedge PN_2 \\
 & & \vee\, \neg(\exists \overline{q}_{\mathcal{A}} : PN_1) \wedge \neg(\exists \overline{q}_{\mathcal{A}} : PN_2) \wedge PN_3 \\
PN_1 & \equiv & \Lambda_j^c \wedge [trans_{\mathcal{A}}]_{\overline{q}_{\mathcal{A}},V} \\
PN_2 & \equiv & \Lambda_j \wedge [trans_{\mathcal{A}}]_{\overline{q}_{\mathcal{A}},V} \\
PN_3 & \equiv & \Lambda_1 \wedge [trans_{\mathcal{A}}]_{\overline{q}_{\mathcal{A}},V}
\end{array}
$$

The invariant I requires $\overline{q}_{\mathcal{A}}$ to encode some element of the label of the root node of the current $\mathcal{D}$-location. Even more, it should be an element of the node numbered j whenever Z_j holds true. For non-stuttering transitions, the transition formula PN requires $\overline{q}_{\mathcal{A}}$ to encode a predecessor of $\overline{q}'_{\mathcal{A}}$ w.r.t. $\mathcal{A}$'s transition relation, chosen (in decreasing order of preference) first among the labels of the children of node j, second as an element of the label of node j, and third as an element of the label of the root. To see that (4.11) indeed defines a prophecy variable, we need to prove

$$\vdash \varphi_{\mathcal{D}}^{j} \rightarrow \Box \exists \overline{q}_{\mathcal{A}} : I \tag{4.12}$$

$$\vdash \varphi_{\mathcal{D}}^{j} \rightarrow \Box[\forall \overline{q}'_{\mathcal{A}} : I' \rightarrow \exists \overline{q}_{\mathcal{A}} : I \wedge PN]_{\overline{q}_{\mathcal{D}},V} \tag{4.13}$$

Neither of these formulas contains flexible quantifiers, and so we may invoke the completeness result of theorem 2.16 and argue semantically. The definition of $\delta^{\mathcal{D}}$ in figure 3.3 immediately implies that the label of the root node of the successor location always contains the set of $\mathcal{A}$-locations reachable from some element of the label set of the root node of the preceding location, and therefore

$$\models [Q_{\mathcal{D},i} \wedge Q'_{\mathcal{D},j} \wedge \delta^{\mathcal{D}}(q_i, q_j)]_{\overline{q}_{\mathcal{D}},V} \rightarrow (\Lambda'_1 \rightarrow \exists \overline{q}_{\mathcal{A}} : \Lambda_1) \qquad (i,j = 1,\ldots,d) \tag{4.14}$$

Observe that (4.14) obviously holds when $\overline{q}_{\mathcal{D}}$ remains unchanged because $\overline{q}_{\mathcal{A}}$ may then be chosen as $\overline{q}'_{\mathcal{A}}$. Inductively, it follows that

$$\models \Box[trans_{\mathcal{D}}]_{\overline{q}_{\mathcal{D}},V} \rightarrow \Box\big((\neg\exists \overline{q}_{\mathcal{A}} : \Lambda_1) \rightarrow \Box(\neg\exists \overline{q}_{\mathcal{A}} : \Lambda_1)\big) \tag{4.15}$$

On the other hand, the condition to color an edge green implies that the source location contains a (non-empty) node numbered j, and therefore we have

$$\models Q_{\mathcal{D},k} \wedge Q'_{\mathcal{D},l} \wedge G_j(q_k, q_l)' \rightarrow \exists \overline{q}_{\mathcal{A}} : \Lambda_j \qquad (k,l = 1,\ldots,d) \tag{4.16}$$

Moreover, condition (K1) of definition 3.3 ensures that $\lambda(q.j) \subseteq \lambda(q.1)$ holds for any location $q \in Q^{\mathcal{D}}$, and therefore

$$\models \Lambda_j \rightarrow \Lambda_1 \tag{4.17}$$

Now, (4.16) and (4.17) imply

$$\models \varphi_{\mathcal{D}}^{j} \to \Box\Diamond\exists\overline{q}_{\mathcal{A}} : \Lambda_1 \tag{4.18}$$

which, together with (4.15) shows

$$\models \varphi_{\mathcal{D}}^{j} \to \Box\exists\overline{q}_{\mathcal{A}} : \Lambda_1 \tag{4.19}$$

On the other hand, Z_j is true only if the current $\mathcal{D}$-location contains a node numbered j, implying

$$\models Z_j \to \exists\overline{q}_{\mathcal{A}} : \Lambda_j \tag{4.20}$$

and therefore (4.12) follows with the help of (4.17) and (4.19).

For the proof of (4.13), we first observe that for every $\mathcal{D}$-transition and every $\mathcal{A}$-location in the label of some node of the successor there is an $\mathcal{A}$-location related by $\delta^{\mathcal{A}}$ in the root of the predecessor, implying

$$\models [trans_{\mathcal{D}}]_{\overline{q}_{\mathcal{D}},V} \wedge \Lambda_1' \to \exists\overline{q}_{\mathcal{A}} : PN_3 \tag{4.21}$$

where, again, the case of stuttering transitions is obvious. Moreover, for every $\mathcal{A}$-location in the label of the j-th node of the successor there is an $\mathcal{A}$-location related by $\delta^{\mathcal{A}}$ in the j-th node of the predecessor, unless the predecessor has no such node or the transition satisfies R_j'. We therefore have

$$\begin{aligned} \models\ & [\delta_{\mathcal{D}}]_{\overline{q}_{\mathcal{D}},V} \wedge \Lambda_j' \wedge \bigwedge_{k,l=0}^{d} [\neg(Q_{\mathcal{D},k} \wedge Q_{\mathcal{D},l}' \wedge R_j(q_k,q_l)')]_{\overline{q}_{\mathcal{D}},V} \wedge \bigvee_{i \in D_j} Q_{\mathcal{D},i} \\ & \to \exists\overline{q}_{\mathcal{A}} : PN_2 \end{aligned} \tag{4.22}$$

From (4.21), (4.22), and the definition of Z_j, (4.13) follows by simple temporal reasoning.

To complete the proof of (4.10), it is enough to show

$$\vdash \underbrace{\varphi_{\mathcal{D}}^{j} \wedge \Box I \wedge \Box[PN]_{\overline{q}_{\mathcal{A}},\overline{q}_{\mathcal{D}},V}}_{S_j} \to \varphi_{\mathcal{A}} \tag{4.23}$$

which, again, is a quantifier-free formula and can therefore be argued semantically. The initialization part $init_{\mathcal{A}}$ follows from $init_{\mathcal{D}}$ and the invariant $\Box\Lambda_1$ because the root of the $\mathcal{D}$-location is initially labelled with the set of all $\mathcal{A}$-states reachable from $q_0^{\mathcal{A}}$. The transition condition $\Box[trans_{\mathcal{A}}]_{\overline{q}_{\mathcal{A}},V}$ is implied by the formula $\Box[PN]_{\overline{q}_{\mathcal{A}},\overline{q}_{\mathcal{D}},V}$. It remains to show that the acceptance condition

$$\begin{aligned} acc_{\mathcal{A}} \ \equiv\ & \vee \bigvee_{k,l=0}^{n} \Box\Diamond\langle Q_{\mathcal{A},k} \wedge Q_{\mathcal{A},l}' \wedge C(q_k,q_l)'\rangle_{\overline{q}_{\mathcal{A}},V} \\ & \vee \bigvee_{k=0}^{n} \Box\Diamond(Q_{\mathcal{A},k} \wedge \delta(q_k,q_k) \wedge C(q_k,q_k)) \end{aligned}$$

is implied by formula S_j. The acceptance condition $acc_{\mathcal{D}}$ implies that "green" transitions (as defined by G_j) are taken infinitely often, whereas "red" transitions (satisfying R_j) are taken only finitely often. Therefore S_j implies $\Diamond\Box Z_j$, and the invariant $\Box I$ implies $\Diamond\Box\Lambda_j$. Now, transitions that satisfy $G_j(q, q')$ are defined such that their target locations q' have a node numbered j without children and that every $\mathcal{A}$-location in $\lambda(q'.j)$ is reached from some $\delta^{\mathcal{A}}$-predecessor in $\lambda(q.j)$, which is either contained in the label of some child node of $q.j$ (and therefore $\overline{q}_{\mathcal{A}}$ can be chosen such that PN_1 holds), or the two $\mathcal{A}$-locations are related by an accepting $\mathcal{A}$-transition (possibly a loop). Therefore,

$$\models S_j \rightarrow (\Box\Diamond(\Lambda_j \wedge \neg\Lambda_j^c) \wedge \Box\Diamond\Lambda_j^c) \vee \bigvee_{k,l=0}^{n} \Box\Diamond\langle Q_{\mathcal{A},k} \wedge Q'_{\mathcal{A},l} \wedge C(q_k, q_l)'\rangle_{\overline{q}_{\mathcal{A}},\overline{q}_{\mathcal{D}},V}$$

The first disjunct on the right-hand side implies that there are infinitely many transitions targetting some $\mathcal{A}$-location contained in some child of a j-node, that is, we must have

$$\Box\Diamond\langle\Lambda_j \wedge \neg\Lambda_j^c \wedge (\Lambda_j^c)'\rangle_{\overline{q}_{\mathcal{A}},\overline{q}_{\mathcal{D}},V}$$

Now, the definition of $\mathcal{D}$ implies that any such transition corresponds to an accepting transition of $\mathcal{A}$, and therefore we conclude

$$\models S_j \rightarrow \bigvee_{k,l=0}^{n} \Box\Diamond\langle Q_{\mathcal{A},k} \wedge Q'_{\mathcal{A},l} \wedge \delta_{\mathcal{A}}(q_k, q_l)' \wedge C(q_k, q_l)'\rangle_{\overline{q}_{\mathcal{A}},\overline{q}_{\mathcal{D}},V} \tag{4.24}$$

From (4.24), it follows that

$$\models S_j \rightarrow acc_{\mathcal{A}}$$

as required: this is obvious in case $\overline{q}_{\mathcal{A}}$ or V actually change infinitely often during the transitions identified by (4.24). On the other hand, propositional reasoning shows

$$\begin{aligned} &\models Q_{\mathcal{A},k} \wedge Q'_{\mathcal{A},l} \wedge \delta_{\mathcal{A}}(q_k, q_l)' \wedge C(q_k, q_l)' \wedge [\mathbf{false}]_{\overline{q}_{\mathcal{A}},V} \\ &\quad \rightarrow Q_{\mathcal{A},k} \wedge \delta_{\mathcal{A}}(q_k, q_k) \wedge C(q_k, q_k) \end{aligned}$$

and therefore (4.24) implies that

$$\bigvee_{k=0}^{n} \Box\Diamond(Q_{\mathcal{A},k} \wedge \delta_{\mathcal{A}}(q_k, q_k) \wedge C(q_k, q_k))$$

must hold, which also implies $acc_{\mathcal{A}}$. Q.E.D.

Theorem 4.9 has shown that the characteristic formula $\chi_{\mathcal{A}}$ describes the language $L(\mathcal{A})$ of a Büchi automaton $\mathcal{A}$ up to V-stuttering equivalence. It should therefore be equivalent to the characteristic formula of the automaton $\mathcal{A}_{sc}^{V}$ considered in theorem 3.11. The following theorem asserts the provability of this equivalence.

Theorem 4.13 *Let $\mathcal{A}$ and $\mathcal{B} = \mathcal{A}_{sc}^{V}$ be Büchi edge automata as in theorem 3.11. Then* $\vdash \chi_{\mathcal{A}} \leftrightarrow \chi_{\mathcal{B}}$.

Proof. As in figure 3.5, let $\mathcal{A} = (Q, q_0, \delta, C)$ and $\mathcal{B} = (\overline{Q}, q_0, \overline{\delta}, \overline{C})$, and assume that $Q = \{q_0, \dots, q_n\}$. The locations of $\mathcal{B}$, other than q_0, are of the form (q, s) for $q \in Q$ and $s \subseteq V$. We denote by $\overline{q}_0^{\mathcal{A}}, \dots, \overline{q}_n^{\mathcal{A}}$ the atomic propositions used to encode the locations of $\mathcal{A}$ in $\varphi_{\mathcal{A}}$, and assume the locations of $\mathcal{B}$ to be represented by propositions $\overline{q}_0^{\mathcal{B}}$ and $\overline{q}_{i,s}^{\mathcal{B}}$, for $i \in \{0, \dots, n\}$ and $s \subseteq V$. Similarly, we write $Q_i^{\mathcal{A}}$ and $Q_{i,s}^{\mathcal{B}}$ to denote the predicate that asserts that q_i, resp. (q_i, s), is the current location of $\mathcal{A}$ or $\mathcal{B}$.

"$\Rightarrow$": For every accepting run of $\mathcal{A}$, a corresponding accepting run of $\mathcal{B}$ is defined by the refinement mapping that maps q_i to (q_i, s) where s is the characteristic predicate of the current state. Formally, we define the state formulas

$$\mathbf{q}_{i,s} \equiv Q_i^{\mathcal{A}} \wedge \chi_s$$

(for $i = 0, \dots, n$ and $s \subseteq V$), let $\mathbf{q}_0 \equiv \mathbf{false}$, and write $F[\mathbf{q}/\overline{q}^{\mathcal{B}}]$ for the formula obtained from F by substituting $\mathbf{q}_0$ for $\overline{q}_0^{\mathcal{B}}$ and $\mathbf{q}_{i,s}$ for $\overline{q}_{i,s}^{\mathcal{B}}$, for all i, s.

Now, the definition of $\mathcal{B}$ is such that

$$\models \chi_s \rightarrow \overline{\delta}(\overline{q}, (q, s)) \qquad \text{and} \qquad \models \chi_s \rightarrow \overline{C}(\overline{q}, (q, s))$$

whenever χ_s implies $\delta(q', q)$ or $C(q', q)$ where q' denotes the $\mathcal{A}$-location underlying $\overline{q}$. It therefore follows that

$$\varphi_{\mathcal{A}} \rightarrow \varphi_{\mathcal{B}}[\mathbf{q}/\overline{q}^{\mathcal{B}}] \tag{4.25}$$

is valid. Because this implication does not contain flexible quantification, we may assume that (4.25) is provable, and the assertion

$$\vdash \chi_{\mathcal{A}} \rightarrow \chi_{\mathcal{B}}$$

follows using the rules ($\exists$-I) and ($\exists$-E).

"$\Leftarrow$": Because a transition of $\mathcal{B}$ in state s can in general only be simulated by a sequence of n (or fewer) $\mathcal{A}$-transitions for state s, the idea is to use stuttering variables. But first, we introduce history variables that record the previous location in the assumed run of $\mathcal{B}$. Formally, we introduce a vector $\overline{h}$ of atomic propositions h_0 and $h_{i,s}$ (for $i = 0, \dots, n$ and $s \subseteq V$) and use ($\exists$-H) to derive

$$\vdash \exists \overline{h} : \underbrace{HI \wedge \Box[HN \wedge (\overline{q}^{\mathcal{B}}, V)' \neq (\overline{q}^{\mathcal{B}}, V)]_{\overline{q}^{\mathcal{B}}, V, \overline{h}}}_{H} \tag{4.26}$$

where HI and HN are defined as

$$HI \equiv h_0 \wedge \bigwedge_{i=0}^{n} \bigwedge_{s \subseteq V} \neg h_{i,s}$$

$$HN \equiv (h_0' \leftrightarrow \overline{q}_0^{\mathcal{B}}) \wedge \bigwedge_{i=0}^{n} \bigwedge_{s \subseteq V} (h_{i,s}' \leftrightarrow \overline{q}_{i,s}^{\mathcal{B}})$$

For $\overline{q} \in \overline{Q}$, we denote by $H_{\overline{q}}$ the predicate that asserts that the vector $\overline{h}$ encodes location $\overline{q}$ of automaton $\mathcal{B}$.

We now tabulate a (finite) function

$$\pi : \overline{Q} \times \overline{Q} \times \{1, \ldots, n\} \to Q$$

such that for any locations $\overline{q}, (q,s) \in \overline{Q}$ where $\overline{\delta}(\overline{q}, (q,s)) \equiv \chi_s$ the following conditions are satisfied (q' denotes the $\mathcal{A}$-location that underlies $\overline{q}$):

- If $\mathcal{A}$ contains a path $q' \stackrel{s^k}{\Rightarrow} q$ then the sequence of $\mathcal{A}$-locations $\langle q', \pi(\overline{q}, (q,s), 1), \ldots, \pi(\overline{q}, (q,s), n)\rangle$ represents such a path of length at most n, possibly followed by stuttering steps; moreover, if $\overline{C}(\overline{q}, (q,s)) \equiv \chi_s$, then that s^n-path contains some accepting transition. Clearly, whenever some such path exists then there must be a path of at most n steps.
- Otherwise, the definition of $\mathcal{B}$ implies that $\overline{q} = (q,s)$, and then $\pi(\overline{q}, (q,s), i) = q$.

Next, we introduce a vector $\overline{b}$ of n Boolean stuttering variables $b_1, \ldots, b_n$ that force any successive changes of $(\overline{q}^{\mathcal{B}}, V)$ to be separated by at least n stuttering steps. Formally, we apply ($\exists$-S) to deduce

$$\exists \overline{b} : \underbrace{BI \wedge \Box[Tick \vee Reset]_{\overline{q}^{\mathcal{B}}, V, \overline{b}} \wedge BL}_{B} \tag{4.27}$$

where BI, $Tick$, $Reset$, and BL are defined as

$$\begin{array}{rcl} B_i & \equiv & \big(\bigwedge_{j=0}^{i-1} \neg b_j\big) \wedge \big(\bigwedge_{j=i}^{n} b_j\big) \\ BI & \equiv & B_1 \\ Tick & \equiv & \bigvee_{i=1}^{n-1} \big(B_i \wedge B'_{i+1} \wedge [\mathbf{false}]_{\overline{q}^{\mathcal{B}}, V}\big) \\ Reset & \equiv & B_n \wedge B'_1 \\ BL & \equiv & \Box\Diamond\langle \mathbf{true} \rangle_{\overline{b}} \end{array}$$

We can now compute the current location of automaton $\mathcal{A}$ by a refinement mapping. Formally, we define the state formulas

$$\mathbf{q}_i \equiv \bigvee_{\pi(\overline{q}, (q_j, s), k) = q_i} (H_{\overline{q}} \wedge \chi_s \wedge Q^{\mathcal{B}}_{j,s} \wedge B_k)$$

for $i = 0, \ldots, n$ (the index of the disjunction ranges over all $\overline{q} \in \overline{Q}$, $s \subseteq V$, $j \in \{0, \ldots, n\}$, and $k \in \{1, \ldots, n\}$). We claim that

$$\models \varphi_{\mathcal{B}} \wedge H \wedge B \to \varphi_{\mathcal{A}}[\mathbf{q}/\overline{q}^{\mathcal{A}}] \tag{4.28}$$

where $F[\mathbf{q}/\overline{q}^{\mathcal{A}}]$ denotes the formula obtained from F by substituting $\mathbf{q}_i$ for $\overline{q}_i^{\mathcal{A}}$, for all i. The assertion

$$\vdash \chi_{\mathcal{B}} \rightarrow \chi_{\mathcal{A}}$$

follows from (4.26), (4.27), and (4.28) by the usual applications of ($\exists$-I) and ($\exists$-E), and the completeness of unquantified propositional TLA* (theorem 2.16).

For the proof of (4.28) we observe that the definition of formula H ensures that the current state always satisfies the transition condition between the previous $\mathcal{B}$-location as recorded by H and the current $\mathcal{B}$-location represented by $\overline{q}^{\mathcal{B}}$, that is,

$$\models \varphi_{\mathcal{B}} \wedge H \rightarrow \Box\big(\bigvee_{\overline{q} \in \overline{q}} \bigvee$$

This observation, together with the conditions on π, implies that the sequence of $\mathcal{A}$-locations identified by $\mathbf{q}$ satisfies the initialization and transition conditions, i.e.

$$\models \varphi_{\mathcal{B}} \wedge H \rightarrow init_{\mathcal{A}}[\mathbf{q}/\overline{q}^{\mathcal{A}}] \wedge \Box[trans_{\mathcal{A}}[\mathbf{q}/\overline{q}^{\mathcal{A}}]]_{\mathbf{q},V}$$

For the acceptance condition there are two cases: it is easy to see that for every run of $\mathcal{B}$ that satisfies

$$\Box\Diamond\langle Q^{\mathcal{B}}_{k,s} \wedge (Q^{\mathcal{B}}_{l,t})' \wedge C^{\mathcal{B}}((q_k,s),(q_l,t))'\rangle_{\overline{q}^{\mathcal{B}},V}$$

the corresponding run of $\mathcal{A}$ represented by $\mathbf{q}$ follows infinitely many accepting (possibly stuttering) transitions, using the definition of π. Otherwise, the run of $\mathcal{B}$ must satisfy

$$\Box\Diamond(Q^{\mathcal{B}}_{k,s} \wedge \delta^{\mathcal{B}}((q_k,s),(q_k,s)) \wedge C^{\mathcal{B}}((q_k,s),(q_k,s))$$

for some k, s, and the sequence $\langle \pi((q_k,s),(q_k,s),1),,\ldots,\pi((q_k,s),(q_k,s),n)\rangle$ is guaranteed to contain some accepting transition. Moreover, the stuttering variables must change value infinitely often, and at least once between any two changes of $\overline{q}^{\mathcal{B}}$, and therefore the run of $\mathcal{A}$ encoded by $\mathbf{q}$ must again contain infinitely many (possibly stuttering) transitions.

Q.E.D.

4.6 Completeness of quantified propositional TLA*

We will now prove the completeness of the axiomatization defined in section 4.4. The main step of the proof is to associate with every formula F a nondeterministic SI_V automaton $\mathcal{A}_F$ such that F and $\chi_{\mathcal{A}_F}$, the characteristic formula of $\mathcal{A}_F$, are provably equivalent. By corollary 4.7, it suffices to consider formulas of quantified propositional TLA.

Theorem 4.14 *Assume that F is a formula of quantified propositional TLA where $At(F) \subseteq V$. There is an SI_V automaton A_F such that*

$$\vdash F \leftrightarrow \chi_{\mathcal{A}_F}$$

The proof of theorem 4.14 proceeds by induction on the structure of F; we now consider each case separately.

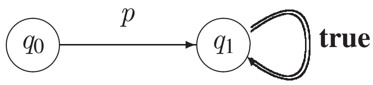


Figure 4.1: Automaton $\mathcal{A}_p$.

Case: $F \equiv p \in V$. The SI_V automaton $\mathcal{A}_p$ for an atomic formula $p \in V$ is defined as in figure 4.1; clearly, $\mathcal{A}_p$ satisfies the conditions (SI0)–(SI3).

Lemma 4.15 $\vdash p \leftrightarrow \chi_{\mathcal{A}_p}$

Proof. The characteristic formula $\chi_{\mathcal{A}_p}$ of automaton $\mathcal{A}_p$ is (up to trivial simplifications)

$$\begin{aligned}\chi_{\mathcal{A}_p} \equiv \exists\, q_0, q_1 : & \wedge\ Q_1 \wedge p \\ & \wedge\ \Box[(Q_0 \wedge Q_1' \wedge p') \vee (Q_1 \wedge Q_1')]_{q_0,q_1,V} \\ & \wedge\ \Box\Diamond\langle Q_1 \wedge Q_1'\rangle_{q_0,q_1,V} \vee \Box\Diamond Q_1\end{aligned}$$

For the proof of $\vdash \chi_{\mathcal{A}_p} \rightarrow p$, it suffices to observe that $\varphi_{\mathcal{A}_p} \rightarrow p$ is a tautology, and therefore the assertion follows by the rule ($\exists$-E).

To prove the other implication, we note that the implication

$$p \rightarrow \varphi_{\mathcal{A}_p}[\mathbf{false}/q_0, \mathbf{true}/q_1] \tag{4.29}$$

is valid: observe that by propositional simplification

$$\begin{aligned}Q_0[\mathbf{false}/q_0, \mathbf{true}/q_1] &\leftrightarrow \mathbf{false} \\ Q_1[\mathbf{false}/q_0, \mathbf{true}/q_1] &\leftrightarrow \mathbf{true}\end{aligned}$$

and therefore it follows that

$$\begin{aligned}\varphi_{\mathcal{A}_p}[\mathbf{false}/q_0, \mathbf{true}/q_1] \leftrightarrow & \wedge\ \mathbf{true} \wedge p \\ & \wedge\ \Box[(\mathbf{false} \wedge \mathbf{true}' \wedge p') \vee (\mathbf{true} \wedge \mathbf{true}')]_{\mathbf{false},\mathbf{true},V} \\ & \wedge\ \Box\Diamond\langle \mathbf{true} \wedge \mathbf{true}'\rangle_{\mathbf{false},\mathbf{true},V} \vee \Box\Diamond\mathbf{true}\end{aligned}$$

From (4.29), the assertion $\vdash p \rightarrow \chi_{\mathcal{A}_p}$ follows by an application of ($\exists$-I). Q.E.D.

Case: $F \equiv G \vee H$. Inductively, we assume that we have already constructed SI_V automata $\mathcal{A}_G = (Q^G, q_0^G, \delta^G, C^G)$ and $\mathcal{A}_H = (Q^H, q_0^H, \delta^H, C^H)$ such that both $\vdash G \leftrightarrow \chi_{\mathcal{A}_G}$ and $\vdash H \leftrightarrow \chi_{\mathcal{A}_H}$; w.l.o.g. we may assume that $Q^G \cap Q^H = \emptyset$. We define the SI_V automaton $\mathcal{A}_F = (Q, q_0, \delta, C)$ as shown in figure 4.2 (where $q_0 \notin Q^G \cup Q^H$ is chosen as a “fresh” location).

It is easy to see that $\mathcal{A}_{G \vee H}$ satisfies the conditions (SI0)–(SI3) provided that $\mathcal{A}_G$ and $\mathcal{A}_H$ do.

$$Q \equiv Q^G \cup Q^H \cup \{q_0\}$$

$$\delta(q,q') \equiv \begin{cases} \delta^G(q,q') & \text{if } q,q' \in Q^G \\ \delta^H(q,q') & \text{if } q,q' \in Q^H \\ \delta^G(q_0^G,q') & \text{if } q = q_0, q' \in Q^G \\ \delta^H(q_0^H,q') & \text{if } q = q_0, q' \in Q^H \\ \textbf{false} & \text{otherwise} \end{cases}$$

$$C(q,q') \equiv \begin{cases} C^G(q,q') & \text{if } q,q' \in Q^G \\ C^H(q,q') & \text{if } q,q' \in Q^H \\ \textbf{false} & \text{otherwise} \end{cases}$$

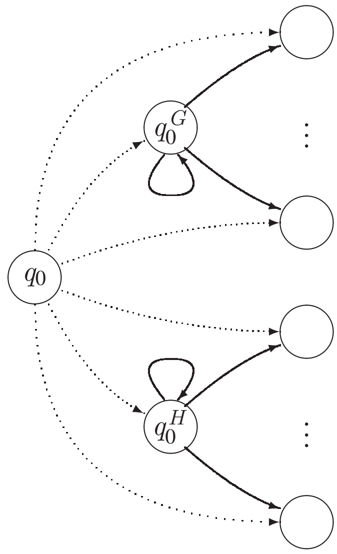


Figure 4.2: Construction of $\mathcal{A}_{G\vee H}$.

Lemma 4.16 $\vdash \chi_{\mathcal{A}_G} \vee \chi_{\mathcal{A}_H} \leftrightarrow \chi_{\mathcal{A}_{G\vee H}}$

Proof. The assertion follows by standard quantifier reasoning from

$$\vdash \varphi_{\mathcal{A}_{G\vee H}} \leftrightarrow \varphi_{\mathcal{A}_G} \vee \varphi_{\mathcal{A}_H} \tag{4.30}$$

observing that $\exists$ commutes over $\vee$. The latter formula is valid because the runs of $\mathcal{A}_{G\vee H}$ are precisely of the form $q_0\rho$ where either $q_0^G\rho$ is a run of $\mathcal{A}_G$ or $q_0^H\rho$ is a run of $\mathcal{A}_H$, since Q^G and Q^H were assumed to be disjoint). Theorem 2.16 ensures that the implication (4.30) can be derived in the axiom system, which completes the proof. Q.E.D.

Case: $F \equiv \Diamond G$. Inductively, we assume that we have already constructed an SI$_V$ automaton $\mathcal{A}_G = (Q^G, q_0^G, \delta^G, C^G)$ such that $\vdash G \leftrightarrow \chi_{\mathcal{A}_G}$. From $\mathcal{A}_G$, we construct $\mathcal{A}_F = (Q, q_0, \delta, C)$ as shown in figure 4.3.

Because the transitions of $\mathcal{A}_{\Diamond G}$ are labelled with the same formulas as those of $\mathcal{A}_G$, except for the additional loop at q_0 labelled with **true**, it is easy to see that $\mathcal{A}_{\Diamond G}$ is an SI$_V$ automaton whenever $\mathcal{A}_G$ is.

The idea of the construction is for $\mathcal{A}_{\Diamond G}$ to nondeterministically guess when G becomes true and then to simulate $\mathcal{A}_G$. Because the loop at q_0 is not an accepting transition, no accepting run of $\mathcal{A}_{\Diamond G}$ can stay at location q_0 forever. More precisely, every accepting run of $\mathcal{A}_{\Diamond G}$ is of the form $q_0 \ldots q_0\rho$ such that $q_0^G\rho$ is an accepting run of $\mathcal{A}_G$.

$$Q \equiv Q^G \cup q_0$$

$$\delta(q,q') \equiv \begin{cases} \delta^G(q,q') & \text{if } q,q' \in Q^G \\ \delta^G(q_0^G,q') & \text{if } q = q_0, q' \in Q^G \\ \textbf{true} & \text{if } q = q' = q_0 \\ \textbf{false} & \text{otherwise} \end{cases}$$

$$C(q,q') \equiv \begin{cases} C^G(q,q') & \text{if } q,q' \in Q^G \\ \textbf{false} & \text{otherwise} \end{cases}$$

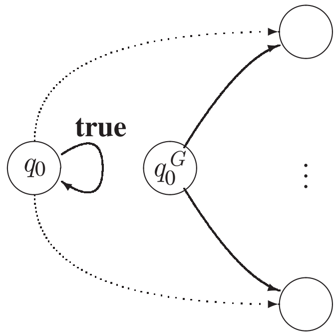


Figure 4.3: Construction of $\mathcal{A}_{\Diamond G}$.

Lemma 4.17 $\vdash \Diamond\chi_{\mathcal{A}_G} \leftrightarrow \chi_{\mathcal{A}_{\Diamond G}}$

Proof. For the proof of the "$\Leftarrow$" direction we note that

$$\models \varphi_{\mathcal{A}_{\Diamond G}} \rightarrow \Diamond\varphi_{\mathcal{A}_G} \tag{4.31}$$

follows from the above observation on the accepting runs of $\mathcal{A}_{\Diamond G}$. Because this formula is quantifier-free, we may assume it to be provable, and then the assertion follows using ($\exists$-I) and ($\exists$-E).

For the proof of the "$\Rightarrow$" direction we first introduce a history variable that indicates whether $\varphi_{\mathcal{A}_G}$ has already been true. Formally, ($\exists$-H) shows

$$\vdash \exists h : \underbrace{(h \leftrightarrow \varphi_{\mathcal{A}_G}) \wedge \Box[h' \leftrightarrow h \vee \circ\varphi_{\mathcal{A}_G} \wedge V' \neq V]_{V,h}}_{H} \tag{4.32}$$

It is then not hard to see that the implication

$$H \wedge \Diamond\varphi_{\mathcal{A}_G} \rightarrow \varphi_{\mathcal{A}_{\Diamond G}}[\neg h/\overline{q}_0] \tag{4.33}$$

is valid, and therefore also provable, by theorem 2.16, and the assertion follows with the help of (4.32) and the rules ($\exists$-I) and ($\exists$-E). Q.E.D.

Case: $F \equiv \Diamond\langle A\rangle_v$. We first transform the action A into disjunctive normal form, which can be written in the form $\bigvee_i (P_i \wedge R_i')$ for state formulas P_i and R_i. Using the equivalence

$$\Diamond\Big\langle \bigvee_i (P_i \wedge R_i')\Big\rangle_v \;\leftrightarrow\; \bigvee_i \Diamond\langle P_i \wedge R_i'\rangle_v$$

we may therefore restrict ourselves to considering the case where $A \equiv P \wedge R'$ for state formulas P and R (and $At(P) \cup At(R) \subseteq V$). We define the automaton $\mathcal{A}_{\Diamond\langle P \wedge R'\rangle_v}$ as in figure 4.4.

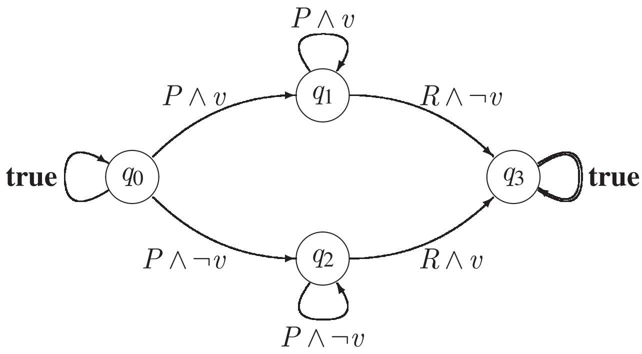


Figure 4.4: Automaton $\mathcal{A}_{\Diamond\langle P \wedge R' \rangle_v}$.

$\mathcal{A}_{\Diamond\langle P \wedge R' \rangle_v}$ is an SI_V automaton. In particular, the atomic proposition v must change along any path from q_0 to q_3, and therefore these states need not be joined by a direct transition.

Lemma 4.18 $\vdash \Diamond\langle P \wedge R' \rangle_v \;\leftrightarrow\; \chi_{\mathcal{A}_{\Diamond\langle P \wedge R' \rangle_v}}$

Proof. We write $\mathcal{A}$ for $\mathcal{A}_{\Diamond\langle P \wedge R' \rangle_v}$. Up to obvious simplifications, the characteristic formula $\chi_{\mathcal{A}}$ is

$$\begin{array}{rl} \chi_{\mathcal{A}} \;\equiv\; \exists \overline{q} : & \wedge\; Q_0 \vee (P \wedge v \wedge Q_1) \vee (P \wedge \neg v \wedge Q_2) \\ & \wedge\; \Box \begin{bmatrix} \vee & Q_0 \wedge Q_0' \\ \vee & (Q_0 \vee Q_1) \wedge P' \wedge v' \wedge Q_1' \\ \vee & (Q_0 \vee Q_2) \wedge P' \wedge \neg v' \wedge Q_2' \\ \vee & Q_1 \wedge R' \wedge \neg v' \wedge Q_3' \\ \vee & Q_2 \wedge R' \wedge v' \wedge Q_3' \\ \vee & Q_3 \wedge Q_3' \end{bmatrix}_{\overline{q}, V} \\ & \wedge\; \Box\Diamond Q_3 \end{array}$$

A straightforward semantic argument shows that the implication

$$\varphi_{\mathcal{A}} \rightarrow \Diamond\langle P \wedge R' \rangle_v \tag{4.34}$$

is valid, because V contains all variables that occur in P, R, and v. By theorem 2.16, this implication is therefore provable, and

$$\vdash \chi_{\mathcal{A}} \rightarrow \Diamond\langle P \wedge R' \rangle_v \tag{4.35}$$

follows with the help of rule ($\exists$-E).

For the proof of the converse implication we use a prophecy variable z that predicts that R is going to be true at the next time v changes value, and that P is going to be true up to that point. Formally, ($\exists$-P) proves

$$\vdash \exists z : \underbrace{\Box[z \leftrightarrow P \land (R' \lor (z' \land (V' = V)))]_{V,z}}_{Z} \tag{4.36}$$

We can now define the current location of $\mathcal{A}$ as a history variable. Formally, ($\exists$-H) shows

$$\exists \overline{q} : \underbrace{QI \land \Box[QN \land (V, z)' \neq (V, z)]_{\overline{q}, V, z}}_{Q} \tag{4.37}$$

where QI and QN are defined as

$$\begin{array}{rcl} QI & \equiv & \lor\ \neg Z \land Q_0 \\ & & \lor\ Z \land v \land Q_1 \\ & & \lor\ Z \land \neg v \land Q_2 \\ QN & \equiv & \lor\ Q_0 \land QI' \\ & & \lor\ (Q_1 \lor Q_2) \land V' \neq V \land Q_3 \\ & & \lor\ Q_1 \land V' = V \land Q_1' \\ & & \lor\ Q_2 \land V' = V \land Q_2' \\ & & \lor\ Q_3 \land Q_3' \end{array}$$

Now, semantic reasoning shows that the implication

$$Q \land Z \land \Diamond\langle P \land R' \rangle_v \rightarrow \varphi_{\mathcal{A}} \tag{4.38}$$

is valid, and therefore also provable, by theorem 2.16. Therefore, the assertion

$$\vdash \Diamond\langle P \land R' \rangle_v \rightarrow \chi_{\mathcal{A}} \tag{4.39}$$

follows with the help of ($\exists$-E), ($\exists$-I), (4.36), and (4.37). Q.E.D.

The use of a prophecy variable in the proof of lemma 4.18 above could be avoided by adding transitions from q_1 and q_2 back to q_0 in case $P \land v$ (resp., $P \land \neg v$) becomes false without $R \land \neg v$ (resp., $R \land v$) becoming true. To satisfy condition (SI2), direct transitions between q_1 and q_2 would then also have to be added.

Case: $F \equiv \exists x : G$. We may inductively assume that we have already constructed an $\mathrm{SI}_{V \cup \{x\}}$ automaton $\mathcal{A}_G$ such that $\vdash G \leftrightarrow \chi_{\mathcal{A}_G}$. We will construct $\mathcal{A}_F$ from $\mathcal{A}_G$ such that an accepting run of $\mathcal{A}_F$ chooses, at each state, a suitable value for x in an accepting run of $\mathcal{A}_G$. However, in view of stuttering invariance, we also have to ensure that $\mathcal{A}_F$ becomes an SI_V automaton, and may therefore have

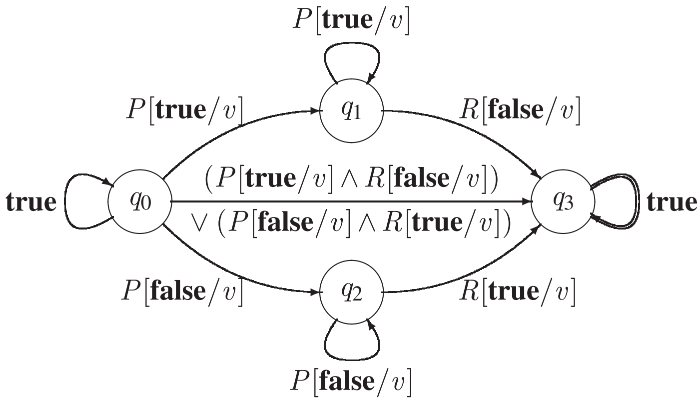


Figure 4.5: Example for the automaton construction for the quantifier case.

to allow for transitions in $\mathcal{A}_F$ that shortcut several $\mathcal{A}_G$-transitions. As an example, consider the formula $F \equiv \exists\, v : \Diamond\langle P \wedge R'\rangle_v$ where P and R are state formulas. The automaton for $\Diamond\langle P \wedge R'\rangle_v$ was already shown in figure 4.4. We intend $\mathcal{A}_F$ to be as shown in figure 4.5. Note the additional transition relating q_0 and q_3. A simple example that justifies its presence is obtained when P and R are atomic propositions: observe that

$$\Diamond(P \wedge R) \rightarrow \exists\, v : \Diamond\langle P \wedge R'\rangle_v$$

is a valid TLA* formula, and must therefore be provable.

Formally, let $\mathcal{A}_G = (Q, q_0, \delta^G, C^G)$ where $Q = \{q_0, \ldots, q_{n-1}\}$, then we define the automaton $\mathcal{A}_F = (Q, q_0, \delta, C)$ where δ and C are defined by the following inductive construction (for all $i, j, k \in \{0, \ldots, n-1\}$):

$$\begin{array}{rcl@{\qquad}rcl}
\delta^0_{ij} & \equiv & \vee\ \delta^G(q_i, q_j)[\mathbf{true}/x] & C^0_{ij} & \equiv & \vee\ C^G(q_i, q_j)[\mathbf{true}/x] \\
 & & \vee\ \delta^G(q_i, q_j)[\mathbf{false}/x] & & & \vee\ C^G(q_i, q_j)[\mathbf{false}/x] \\
\delta^{k+1}_{ij} & \equiv & \delta^k_{ij} \vee (\delta^k_{ik} \wedge \delta^k_{kj}) & C^{k+1}_{ij} & \equiv & C^k_{ij} \vee (\delta^k_{ik} \wedge C^k_{kj}) \\
 & & & & & \quad \vee\ (C^k_{ik} \wedge \delta^k_{kj}) \\
\delta(q_i, q_j) & \equiv & \delta^n_{ij} & C(q_i, q_j) & \equiv & C^n_{ij}
\end{array}$$

Lemma 4.19 *If $\mathcal{A}_G$ is an $SI_{V\cup\{x\}}$ automaton then $\mathcal{A}_F$ is an SI_V automaton.*

Proof. Because we assume $\mathcal{A}_G$ to be an $\mathrm{SI}_{V\cup\{x\}}$ automaton, we have $At(\delta^G(q_i, q_j)) \subseteq V \cup \{x\}$, and similarly for $C^G(q_i, q_j)$. The definitions of δ^0_{ij} and of C^0_{ij} therefore ensure that $At(\delta^0_{ij}) \subseteq V$ and $At(C^0_{ij}) \subseteq V$ for all i, j. Inductively, it follows that $At(\delta^k_{ij}) \subseteq V$ and $At(C^k_{ij}) \subseteq V$ for all i, j, k, and therefore also $At(\delta(q_i, q_j)) \subseteq V$ and $At(C(q_i, q_j)) \subseteq V$. This shows that condition (SI0) holds of $\mathcal{A}_F$.

Observe that the definition of δ_{ij}^k immediately implies

$$\models \delta_{ij}^k \to \delta_{ij}^l \quad \text{for } k \le l \tag{4.40}$$

From the definition of $\delta(q_i, q_j)$ and observation (4.40) above, condition (SI1) follows from the more general claim

$$\models \delta_{ij}^k \to \delta_{jj}^0 \tag{4.41}$$

which is shown by induction on k. The base case holds because δ^G is assumed to satisfy (SI1), implying that

$$\models \delta^G(q_i, q_j)[b/x] \to \delta^G(q_j, q_j)[b/x]$$

holds for both $b \equiv$ **true** and $b \equiv$ **false**. For the induction step, we distinguish two cases according to the definition of δ_{ij}^{k+1}. First, the induction hypothesis immediately yields $\models \delta_{ij}^k \to \delta_{jj}^0$. Second, again by the induction hypothesis we have $\models \delta_{kj}^k \to \delta_{jj}^0$ and therefore also $\models \delta_{ik}^k \wedge \delta_{kj}^k \to \delta_{jj}^0$, which implies the assertion.

To prove condition (SI2), we show that

$$\models \delta_{ij}^k \wedge \delta_{jl}^k \to \delta_{il}^k \tag{4.42}$$

holds whenever $j < k$; again we use induction on k. The base case is trivial. For the inductive step, we assume that $j < k+1$. The definition of δ_{ij}^{k+1} and simple boolean manipulations show that

$$\begin{aligned}\models \delta_{ij}^{k+1} \wedge \delta_{jl}^{k+1} \;\leftrightarrow\; & \vee\, \delta_{ij}^k \wedge \delta_{jl}^k \\ & \vee\, \delta_{ij}^k \wedge \delta_{jk}^k \wedge \delta_{kl}^k \\ & \vee\, \delta_{ik}^k \wedge \delta_{kj}^k \wedge \delta_{jl}^k \\ & \vee\, \delta_{ik}^k \wedge \delta_{kj}^k \wedge \delta_{jk}^k \wedge \delta_{kl}^k\end{aligned}$$

and we prove that each disjunct on the right-hand side implies δ_{il}^{k+1}.

If $j < k$ then, by the induction hypothesis, the first disjunct implies δ_{il}^k, and therefore also δ_{il}^{k+1}, using (4.40). For $j = k$ the formula implies $\delta_{ik}^k \wedge \delta_{kl}^k$ and therefore again δ_{il}^k by induction hypothesis.

Considering the second disjunct, if $j < k$ then $\delta_{ij}^k \wedge \delta_{jk}^k$ implies δ_{ik}^k by the induction hypothesis, which together with δ_{kl}^k implies δ_{il}^{k+1} by the definition of δ_{il}^{k+1}. If $j = k$, we immediately have δ_{ik}^k, and therefore again δ_{il}^{k+1}.

The case of the third disjunct is symmetrical.

Finally, the fourth disjunct implies both δ_{ik}^k and δ_{kl}^k and therefore, by definition, again δ_{il}^{k+1}.

The proof of condition (SI3) is similar, using the definitions of both C_{ij}^k and δ_{ij}^k, and relying on the just established validity of (4.42). Q.E.D.

Lemma 4.20 $\vdash (\exists x : \chi_{\mathcal{A}_G}) \leftrightarrow \chi_{\mathcal{A}_F}$

Proof. For the proof of the implication "$\Rightarrow$", it suffices to show

$$\vdash \varphi_{\mathcal{A}_G} \rightarrow \varphi_{\mathcal{A}_F} \tag{4.43}$$

using the quantifier rules ($\exists$-I) and ($\exists$-E) and the induction hypothesis that $\vdash G \leftrightarrow \chi_{\mathcal{A}_G}$. But the above assertion (which essentially amounts to showing that every successful run of $\mathcal{A}_G$ is a successful run of $\mathcal{A}_F$) follows immediately from the definitions of $\varphi_{\mathcal{A}_G}$ and $\varphi_{\mathcal{A}_F}$, observing that

$$\vdash \delta^G(q_i, q_j) \rightarrow \delta^G(q_i, q_j)[\mathbf{true}/x] \vee \delta^G(q_i, q_j)[\mathbf{false}/x]$$

where the right-hand side is just δ^0_{ij}, and that the definition of $\delta(q_i, q_j)$ and δ^k_{ij} immediately imply $\vdash \delta^0_{ij} \rightarrow \delta(q_i, q_j)$. Similarly, we have

$$\vdash C^G(q_i, q_j) \rightarrow C^G(q_i, q_j)[\mathbf{true}/x] \vee C^G(q_i, q_j)[\mathbf{false}/x]$$

and therefore $\vdash C^G(q_i, q_j) \rightarrow C(q_i, q_j)$, which proves that every run accepted by $\mathcal{A}_G$ is also accepted by $\mathcal{A}_F$.

For the proof of the implication "$\Leftarrow$", we have to simulate an accepting run of $\mathcal{A}_F$ over a temporal structure σ by an accepting run of $\mathcal{A}_G$ over some structure τ such that $\tau \approx_x \sigma$. Because $\mathcal{A}_F$ has more transitions than $\mathcal{A}_G$, we may have to introduce additional stuttering. The idea is to simulate a single transition of $\mathcal{A}_F$ by $n-1$ transitions of $\mathcal{A}_G$: given a transition from q_i to q_j in $\mathcal{A}_F$ and a valuation ξ that satisfies $\delta(q_i, q_j)$, we can find a path in $\mathcal{A}_G$ whose transition labels can all be satisfied by ξ and suitable choices for x (these choices may be different along the path). Because $\mathcal{A}_G$ has n locations, we may always choose a path that has at most $n-1$ transitions; on the other hand, we can extend a shorter path by loops at the target location, because $\mathcal{A}_G$ is an $\mathrm{SI}_{V \cup \{x\}}$ automaton and therefore satisfies condition (SI1). More precisely, we tabulate a finite function

$$\pi : \{0, \dots, n-1\} \times \{0, \dots, n-1\} \times 2^V \times \{0, \dots, n\} \rightarrow \{0, \dots, n-1\}$$

such that for all $q_i, q_j \in Q$ and all $s \subseteq V$ such that $\chi_s \rightarrow \delta(q_i, q_j)$ the following conditions are satisfied:

- $\pi(i, j, s, 0) = i$,
- $\pi(i, j, s, n) = j$,
- $\chi_s \rightarrow \delta^0_{\pi(i,j,s,l)\pi(i,j,s,l+1)}$ holds for all $l \in \{0, \dots, n-1\}$, and
- if $\chi_s \rightarrow C(q_i, q_j)$ then $\chi_s \rightarrow C^0_{\pi(i,j,s,l)\pi(i,j,s,l+1)}$ for some $l \in \{0, \dots, n-1\}$.

The existence of such a function π follows easily from the definitions of $\delta(q_i, q_j)$ and $C(q_i, q_j)$.

We start by introducing a vector $\overline{h}$ of history variables $h_0, \ldots, h_{n-1}$ that record the previous location in the run of $\mathcal{A}_F$: by ($\exists$-H) we obtain

$$\vdash \exists \overline{h} : \underbrace{HI \wedge \Box[HN \wedge (\overline{q}, V)' \neq (\overline{q}, V)]_{\overline{q}, V, \overline{h}}}_{H} \tag{4.44}$$

where HI and HN are defined as

$$\begin{array}{rcl} HI & \equiv & h_0 \wedge \bigwedge_{i=1}^{n-1} \neg h_i \\ HN & \equiv & \bigwedge_{i=0}^{n-1} (h_i' \leftrightarrow \overline{q}_i) \end{array}$$

For $i = 0, \ldots, n-1$, we denote by H_i the predicate that asserts that the vector $\overline{h}$ encodes location q_i.

We now introduce a vector $\overline{b}$ of $n-1$ Boolean stuttering variables $b_1, \ldots, b_{n-1}$ that force any successive changes of $(\overline{q}, V)$ to be separated by at least $n-1$ stuttering steps. Formally, we apply ($\exists$-S) to deduce

$$\exists \overline{b} : \underbrace{BI \wedge \Box[Tick \vee Reset]_{\overline{q}, V, \overline{b}} \wedge BL}_{B} \tag{4.45}$$

where BI, $Tick$, $Reset$, and BL are defined as

$$\begin{array}{rcl} B_i & \equiv & (\bigwedge_{j=0}^{i-1} \neg b_j) \wedge (\bigwedge_{j=i}^{n-1} b_j) \\ BI & \equiv & B_1 \\ Tick & \equiv & \bigvee_{i=1}^{n-2} \left(B_i \wedge B_{i+1}' \wedge [\mathbf{false}]_{\overline{q}, V}\right) \\ Reset & \equiv & B_{n-1} \wedge B_1' \\ BL & \equiv & \Box\Diamond\langle \mathbf{true} \rangle_{\overline{b}} \end{array}$$

Much as in the proof of theorem 4.13 we can now compute the current location of automaton $\mathcal{A}_G$ by a refinement mapping. Formally, we define the state formulas

$$\mathbf{q}_l \equiv \bigvee_{\pi(i,j,s,k)=l} (H_i \wedge Q_j \wedge \chi_s \wedge B_k)$$

for $l = 0, \ldots, n-1$ (the index of the disjunction ranges over all $i, j \in \{0, \ldots, n-1\}$, all $k \in \{1, \ldots, n-1\}$, and all $s \subseteq V$). We denote by $\mathbf{Q}_l$ the formula

$$\mathbf{Q}_l \equiv \mathbf{q}_l \wedge \bigwedge_{k \neq l} \neg \mathbf{q}_k$$

It remains to choose suitable values for x at each state. The conditions on the function π imply that $\delta^G(q_{\pi(q_i,q_j,s,k)}, q_{\pi(q_i,q_j,s,k+1)})[b/x]$ holds at each step, for $b \equiv$ **true** or for $b \equiv$ **false**. In case both formulas happen to hold, we should choose the value such that accepting transitions are preferred. Formally, we apply ($\exists$-H) to introduce a history variable **x** as follows:

$$\vdash \Box(\bigvee_{l=0}^{n-1} \mathbf{Q}_l) \rightarrow \boldsymbol{\exists}\mathbf{x} : \underbrace{XI \wedge \Box[XN \wedge (\overline{q}, V, b)' \neq (\overline{q}, V, b)]_{\overline{q},V,b,x}}_{X} \tag{4.46}$$

where XI and XN are defined as

$$\begin{aligned} XI &\equiv \bigvee_{l=0}^{n-1} \mathbf{Q}_l \wedge (\mathbf{x} \leftrightarrow \delta^G(q_0, q_l)[\mathbf{true}/x]) \\ XN &\equiv \bigvee_{l,m=0}^{n-1} \mathbf{Q}_l \wedge \mathbf{Q}'_m \wedge (\mathbf{x}' \leftrightarrow X(l,m)') \\ X(l,m) &\equiv \wedge\ \delta^G(q_l, q_m)[\mathbf{true}/x] \\ &\qquad \wedge\ \delta^G(q_l, q_m)[\mathbf{false}/x] \wedge C^G(q_l, q_m)[\mathbf{false}/x] \rightarrow C^G(q_l, q_m)[\mathbf{true}/x] \end{aligned}$$

The definition of **x** ensures that **x** is true when the predicate that labels the current transition of $\mathcal{A}_G$ holds when **true** is substituted for x and either the acceptance condition for that transition holds when **true** is substituted for x or the choice of $\mathbf{x} \equiv$ **false** would not make the transition accepting for G. Assembling the bits and pieces, it follows that

$$\models \varphi_{\mathcal{A}_G} \wedge H \wedge B \wedge X \rightarrow \varphi_{\mathcal{A}_F}[\mathbf{q}/\overline{q}, \mathbf{x}/x] \tag{4.47}$$

from which the assertion follows by theorem 2.16, the rules ($\exists$-I) and ($\exists$-E), and assertions (4.44)–(4.46) above. Q.E.D.

Case: $F \equiv \neg G$. The case of negation has traditionally been the most difficult one. We will reuse the results assembled in sections 4.5 for the construction of an SI_V automaton $\mathcal{A}_F$ from the SI_V automaton $\mathcal{A}_G$, which we inductively assume to exist. The construction proceeds in four steps as follows:

1. Construct a deterministic Rabin automaton $\mathcal{R} = (Q^R, q_0^R, \delta^R, C^R)$ with acceptance condition $C^R = \{(R_1, G_1), \ldots, (R_m, G_m)\}$ from $\mathcal{A}_G$, using the construction of figure 3.3.

2. Construct a deterministic Streett automaton $\mathcal{S}$ from $\mathcal{R}$ by complementing the acceptance condition of $\mathcal{R}$. Formally, $\mathcal{S} = (Q^R, q_0^R, \delta^R, C^S)$ where $C^S = \{(G_1, R_1), \ldots, (G_m, R_m)\}$.

3. Construct a nondeterministic Büchi automaton $\mathcal{B}$ equivalent to $\mathcal{S}$ according to lemma 3.2.

4. Construct the SI_V automaton $\mathcal{A}_F = \mathcal{B}_{sc}^V$ from $\mathcal{B}$ as in figure 3.5.

Lemma 4.21 *If $\mathcal{L}(\mathcal{A}_G)$ is closed under V-stuttering equivalence then* $\vdash \neg\chi_{\mathcal{A}_G} \leftrightarrow \chi_{\mathcal{A}_F}$

Proof. Theorems 4.12, 4.11, and 4.13 already imply that

$$\vdash \chi_{\mathcal{A}_G} \leftrightarrow \chi_{\mathcal{R}}$$
$$\vdash \chi_{\mathcal{S}} \leftrightarrow \chi_{\mathcal{B}}$$
$$\vdash \chi_{\mathcal{B}} \leftrightarrow \chi_{\mathcal{A}_F}$$

It therefore remains to establish the correctness of step 2 and show

$$\vdash \chi_{\mathcal{S}} \leftrightarrow \neg\chi_{\mathcal{R}} \tag{4.48}$$

We prove (4.48) by showing the two following assertions:

$$\vdash \chi_{\mathcal{S}} \vee \chi_{\mathcal{R}} \tag{4.49}$$
$$\vdash \neg(\chi_{\mathcal{S}} \wedge \chi_{\mathcal{R}}) \tag{4.50}$$

Assertion (4.50) follows by standard quantifier rules from

$$\vdash \neg\big(\varphi_{\mathcal{S}}[\overline{q}_{\mathcal{S}}/\overline{q}] \wedge \varphi_{\mathcal{R}}[\overline{q}_{\mathcal{R}}/\overline{q}]\big) \tag{4.51}$$

where $\overline{q}_{\mathcal{S}}$ and $\overline{q}_{\mathcal{R}}$ are distinct tuples of "fresh" variables. Using the completeness theorem 2.16, we may prove assertion (4.51) semantically. So assume that $\sigma = s_0 s_1 \ldots$ is a temporal structure that satisfies both $\varphi_{\mathcal{S}}[\overline{q}_{\mathcal{S}}/\overline{q}]$ and $\varphi_{\mathcal{R}}[\overline{q}_{\mathcal{R}}/\overline{q}]$. Let $\mathbf{i} = i_0, i_1, \ldots$ and $\mathbf{j} = j_0, j_1, \ldots$ be the index positions where $i_0 = j_0 = 0$ and the positions i_{k+1} and j_{k+1} are such that $(\overline{q}_{\mathcal{R}}, V)$, respectively $(\overline{q}_{\mathcal{S}}, V)$, change value; observe that the definitions of $\varphi_{\mathcal{R}}$ and $\varphi_{\mathcal{S}}$ ensure that both $\mathbf{i}$ and $\mathbf{j}$ are infinite sequences. Now consider the temporal structures $\alpha = s_{i_0} s_{i_1} \ldots$ and $\beta = s_{j_0} s_{j_1} \ldots$. From the definition of $\varphi_{\mathcal{R}}$ and $\varphi_{\mathcal{S}}$ it follows easily that $\alpha \in \mathcal{L}(\mathcal{R})$ and $\beta \in \mathcal{L}(\mathcal{S})$. Now, theorem 3.6 implies that $\mathcal{L}(\mathcal{R}) = \mathcal{L}(\mathcal{A}_G)$, so $\alpha \in \mathcal{L}(\mathcal{A}_G)$. On the other hand, the definitions of $\mathbf{i}$ and $\mathbf{j}$ ensure that $\alpha \simeq_V \beta$, and by the assumed closure of $\mathcal{L}(\mathcal{A}_G)$ under V-stuttering equivalence we must have $\beta \in \mathcal{L}(\mathcal{A}_G) = \mathcal{L}(\mathcal{R})$. But since $\mathcal{R}$ and $\mathcal{S}$ are deterministic and have the same initial location and transition conditions, there is a unique run of $\mathcal{R}$ and $\mathcal{S}$ over β, which cannot satisfy both C^R and C^S, and a contradiction is reached.

Intuitively, (4.49) holds because the fact that $\mathcal{R}$ and $\mathcal{S}$ are deterministic automata with the same transition relation ensures that there is a (unique) run of $\mathcal{R}$ and $\mathcal{S}$ over any given temporal structure, which must be accepting for either $\mathcal{R}$ or $\mathcal{S}$. To make this argument formal, we define such a run by a history variable, using a stuttering variable to ensure that the run we obtain is infinite. We thus start by introducing a Boolean stuttering variable s, distinct from the propositions in V and $\overline{q}_R$, that ticks infinitely often: ($\exists$-S) proves

$$\vdash \exists\, s : \Box\Diamond\langle \mathbf{true} \rangle_s \tag{4.52}$$

We now construct a run of $\mathcal{R}$ (and $\mathcal{S}$) as a history variable. Formally, ($\exists$-H) can be used to prove

$$\vdash \exists \overline{q}_R : init_R \wedge \Box[trans_R \wedge (V,s)' \neq (V,s)]_{\overline{q}_R,V,s} \tag{4.53}$$

because the transition relation δ^R is total by definition. Moreover, since $\mathcal{R}$ is loop-free it follows that

$$\models init_R \wedge \Box[trans_R \wedge (V,s)' \neq (V,s)]_{\overline{q}_R,V,s} \wedge \Box\Diamond\langle\mathbf{true}\rangle_s \rightarrow \Box\Diamond\langle\mathbf{true}\rangle_{\overline{q}_R,V} \tag{4.54}$$

By propositional temporal logic, it follows that

$$\models init_R \wedge \Box[trans_R \wedge (V,s)' \neq (V,s)]_{\overline{q}_R,V,s} \wedge \Box\Diamond\langle\mathbf{true}\rangle_{\overline{q}_R,V} \rightarrow acc_{\mathcal{R}} \vee acc_{\mathcal{S}} \tag{4.55}$$

and the completeness theorem 2.16 ensures that the implications in (4.54) and (4.55) are provable. Now, assertion (4.49) is obtained from (4.52)–(4.55) and ordinary quantifier reasoning. Q.E.D.

This completes the proof of theorem 4.14. As an immediate corollary, we obtain that the existential fragment of qpTLA* is expressively complete.

Corollary 4.22 *Every formula F of* qpTLA* *is provably equivalent to some formula of the shape $\exists x : G$ where x is a list of propositions and G is a* pTLA* *formula (in particular, G does not contain quantification over atomic propositions).*

As a second corollary to theorem 4.14, coupled with the well-known decidability of the emptiness problem for Büchi automata (which is easily transferred to Büchi edge automata), we obtain a decidability result for qpTLA*. Because every occurrence of a negation symbol may lead to an exponential increase in the number of automata locations, this procedure is of non-elementary complexity. This is to be expected because it follows from results by Meyer and Stockmeyer [76, 92] that the decision problem for QPTL admits no algorithm of elementary complexity.

Corollary 4.23 *The satisfiability problem for* qpTLA* *is decidable.*

We are now in a position to prove the desired completeness theorem for quantified propositional TLA*.

Theorem 4.24 *For any formula F of* qpTLA*, *$\models F$ implies $\vdash F$.*

Proof. Suppose that F is a valid qpTLA* formula, so $\neg F$ is unsatisfiable. By theorem 4.14, we have

$$\vdash \neg F \leftrightarrow \chi_{\mathcal{A}_{\neg F}} \tag{4.56}$$

and since our rules are sound (theorem 4.8), $\chi_{\mathcal{A}_{\neg F}}$ is therefore also unsatisfiable. In other words, we have

$$\models \neg \exists\!\!\exists\, \overline{q} : \varphi_{\mathcal{A}_{\neg F}} \tag{4.57}$$

and therefore also

$$\models \neg \varphi_{\mathcal{A}_{\neg F}} \tag{4.58}$$

The formula in (4.58) does not contain flexible quantification, and therefore theorem 2.16 implies

$$\vdash \neg \varphi_{\mathcal{A}_{\neg F}} \tag{4.59}$$

By rule ($\exists\!\!\exists$-E) and propositional reasoning we obtain

$$\vdash \neg \exists\!\!\exists\, \overline{q} : \varphi_{\mathcal{A}_{\neg F}} \tag{4.60}$$

and the desired result $\vdash F$ follows from (4.56), (4.60), and propositional logic. Q.E.D.

4.7 First-order TLA*

The extension of propositional TLA* to a full first-order temporal logic is essentially straightforward. Unlike in the propositional fragment, but following Lamport [62], we will include primed flexible variables in the base syntax of first-order TLA* rather than introduce them as a derived notion. This greatly simplifies the notation because otherwise (rigid) quantification would be necessary to refer to the value of a variable at the successor state. Also note that the logic defined in this section is untyped, following both standard mathematical logic and Lamport's definition of TLA, although a multisorted variant of TLA* could be defined in the usual manner.

Assume given a language $\mathcal{L}$ of first-order logic that defines sets $\mathcal{F}$ and $\mathcal{P}$ of function and predicate symbols, each equipped with an arity $n \geq 0$. Assume further given a set $\mathcal{X} = \mathcal{X}_r \cup \mathcal{X}_f$ of variables, partitioned into disjoint sets $\mathcal{X}_r$ and $\mathcal{X}_f$ of *rigid* and *flexible* variables. (We assume that $\mathcal{X}$ does not contain symbols of the form x'.) The terms and formulas of TLA* are defined much as in classical first-order logic; we also define *transition terms* that may contain primed flexible variables and are used in the definition of pre-formulas.

Definition 4.25 (terms) *The* terms *of* TLA* *are inductively defined as follows.*

1. *Every variable $x \in X$ is a term.*

2. *If $t_1, \ldots, t_n$ are terms and f is an n-ary function symbol then $f(t_1, \ldots, t_n)$ is a term. For $n = 0$ we usually write f instead of $f()$.*

The transition terms *of* TLA* *are defined similarly:*

1. *Every variable $x \in X$ is a transition term. For a flexible variable $v \in X_f$, also v' is an transition term.*

2. *If $t_1, \ldots, t_n$ are transition terms and f is an n-ary function symbol then $f(t_1, \ldots, t_n)$ is a transition term.*

For a term t, we denote by t' the transition term obtained from t by replacing every flexible variable v in t by v'.

Definition 4.26 (formulas) *Formulas and pre-formulas of* TLA* *are inductively defined as follows.*

1. **false** *is a formula.*

2. *If $t_1, \ldots, t_n$ are terms and P is an n-ary predicate symbol then $P(t_1, \ldots, t_n)$ is a formula. If t_1 and t_2 are terms then $t_1 = t_2$ is a formula.*

3. *If F, G are formulas then $F \rightarrow G$ and $\Box F$ are formulas.*

4. *If A is a pre-formula and t is a term then $\Box[A]_t$ is a formula.*

5. *If F is a formula and $x \in X_r$ is a rigid variable then $\exists x : F$ is a formula.*

6. *If F is a formula and $v \in X_f$ is a flexible variable then $\boldsymbol{\exists}\, v : F$ is a formula.*

7. *If F is a formula then F and $\circ F$ are pre-formulas.*

8. *If $t_1, \ldots, t_n$ are transition terms and P is an n-ary predicate symbol then $P(t_1, \ldots, t_n)$ is a pre-formula. If t_1 and t_2 are transition terms then $t_1 = t_2$ is a pre-formula.*

9. *If A, B are pre-formulas then $A \rightarrow B$ is a pre-formula.*

10. *If A is a pre-formula and $x \in X_r$ is a rigid variable then $\exists x : A$ is a pre-formula.*

Free and bound occurrences of variables, and the sets $FFV(F)$ and $FRV(F)$ of free (flexible or rigid) variables that occur in (pre-)formula F are defined in the standard way. Note that $\exists\!\!\exists$ binds unprimed as well as primed occurrences of the quantified flexible variable.

We use notation analogous to that introduced for (quantified) propositional TLA*. For example, $\Box[F]_{t_1,\ldots,t_n}$ denotes the formula

$$\Box[F]_{t_1} \wedge \ldots \wedge \Box[F]_{t_n}$$

We now write $\Box[A]_F$, for a formula F and a pre-formula A, to denote the formula $\Box[A]_{FFV(F)}$. Substitution is now defined at the level of terms, so $F[t/x]$ denotes the (pre-)formula obtained from the (pre-)formula F by substituting the term t for all free occurrences of the (rigid or flexible) variable x

State formulas are those formulas that do not contain any temporal operators. Similarly, *action formulas* (actions for short) are those pre-formulas that are built from (possibly primed) variables, function and predicate symbols (including $=$), and first-order connectives, but that do not contain any temporal connectives. TLA is the sublogic of TLA* obtained by restricting pre-formulas to be actions. We sometimes write P', for a state formula P, to denote the action formula obtained from P by replacing every free flexible variable v in P by its primed counterpart v'. If A is an action we write

$$\text{ENABLED } A \quad \equiv \quad \exists v_1', \ldots, v_n' : A$$

to denote the state formula obtained from A by existentially quantifying over all free primed flexible variables $v_1', \ldots, v_n'$ in A.[1] Furthermore, if t is a term (or finite set of terms), we write

$$\begin{aligned} \mathrm{WF}_t(A) &\equiv \Diamond\Box\text{ENABLED } \langle A\rangle_t \rightarrow \Box\Diamond\langle A\rangle_t \\ \mathrm{SF}_t(A) &\equiv \Box\Diamond\text{ENABLED } \langle A\rangle_t \rightarrow \Box\Diamond\langle A\rangle_t \end{aligned}$$

The semantics of TLA* is defined relative to a first-order interpretation I that provides a universe $|I|$ and, for all function and predicate symbols $f \in \mathcal{F}$ and $P \in \mathcal{P}$, interpretations f^I and P^I, which are functions and relations of suitable arity. A *state* is now a valuation $s : \mathcal{X}_f \rightarrow |I|$ of the flexible variables. A *behavior* is an infinite sequence $\sigma = s_0 s_1 \ldots$ of states.

For a set $X \subseteq \mathcal{X}_f$ of flexible variables, the relations $=_X$ and $\simeq_X$ are defined as in definitions 2.3. Similarly, for $x \in \mathcal{X}_f$ the relations $=_x$ and $\approx_x$ are defined as in definition 4.2. We note that propositions 2.4 and 4.3 equally hold for these relations.

[1] Note that we use rigid quantification over $v_1', \ldots, v_n'$. To be truly formal, we should replace all primed occurrences of flexible variables by fresh rigid variables or define the syntax as in [7].

Definition 4.27 (term semantics) *Given an interpretation I and a valuation of the rigid variables $\alpha : \mathcal{X}_r \rightarrow |I|$, the semantics $s[\![t]\!]^{I,\alpha}$ of a transition term t at states s and u is inductively defined as follows:*

$$\begin{array}{rcll}
s[\![x]\!]^{I,\alpha}u & = & \alpha(x) & \textit{for } x \in \mathcal{X}_r \\
s[\![v]\!]^{I,\alpha}u & = & s(v) & \textit{for } v \in \mathcal{X}_f \\
s[\![v']\!]^{I,\alpha}u & = & u(v) & \textit{for } v \in \mathcal{X}_f \\
s[\![f(t_1,\ldots,t_n)]\!]^{I,\alpha}u & = & f^I(s[\![t_1]\!]^{I,\alpha}u,\ldots,s[\![t_1]\!]^{I,\alpha}u) &
\end{array}$$

Because every term is a transition term, definition 4.27 also defines the semantics of terms. Clearly, the second state u is irrelevant for the semantics of terms; we will therefore often write $s[\![t]\!]^{I,\alpha}$ when t is a term.

Definition 4.28 (formula semantics) *The semantics of (pre-)formulas, relative to an interpretation I, a valuation $\alpha : \mathcal{X}_r \rightarrow |I|$, and a behavior $\sigma = s_0 s_1 \ldots$ is inductively defined as follows:*

$$\begin{array}{rll}
I,\sigma,\alpha \not\approx \mathbf{false} & & \\
I,\sigma,\alpha \approx P(t_1,\ldots,t_n) & \textit{iff} & (s_0[\![t_1]\!]^{I,\alpha}s_1,\ldots,s_0[\![t_n]\!]^{I,\alpha}s_1) \in P^I \\
I,\sigma,\alpha \approx t_1 = t_2 & \textit{iff} & s_0[\![t_1]\!]^{I,\alpha}s_1 = s_0[\![t_2]\!]^{I,\alpha}s_1 \\
I,\sigma,\alpha \approx F \rightarrow G & \textit{iff} & I,\sigma,\alpha \approx F \textit{ implies } I,\sigma,\alpha \approx G \\
I,\sigma,\alpha \approx \Box F & \textit{iff} & I,\sigma[i..],\alpha \approx F \textit{ holds for all } i \in \mathbb{N} \\
I,\sigma,\alpha \approx \Box[A]_t & \textit{iff} & \textit{for all } i \in \mathbb{N}, s_i[\![t]\!]^{I,\alpha} = s_{i+1}[\![t]\!]^{I,\alpha} \textit{ or } \sigma[i..] \approx A
\end{array}$$

$$\begin{array}{rll}
I,\sigma,\alpha \approx \exists x : F & \textit{iff} & I,\sigma,\beta \approx F \textit{ for some valuation } \beta \textit{ such that} \\
& & \beta(y) = \alpha(y) \textit{ for all } y \not\equiv x \\
I,\sigma,\alpha \approx \boldsymbol{\exists}\, v : F & \textit{iff} & I,\tau,\alpha \approx F \textit{ for some behavior } \tau \approx_v \sigma \\
I,\sigma,\alpha \approx \circ F & \textit{iff} & I,\sigma[1..],\alpha \approx F
\end{array}$$

If F is a formula, we usually write $I,\sigma,\alpha \models F$ instead of $I,\sigma,\alpha \approx F$.

In the following, we will always assume that the first-order interpretation is understood and suppress I. For closed formulas we also drop the rigid valuation α. Finally, if P is a state formula, the semantics only depends on the initial state s of σ, and we often write $s[\![P]\!]^{\alpha}$ to emphasize the similarity with the term semantics. Similarly, we write $s[\![A]\!]^{\alpha}u$ to denote the semantics of an action formula A at states s and u.

For example, the semantics ensures that P' and $\circ P$ are equivalent when P is a state formula, and that $s[\![P']\!]^{\alpha}u$ holds iff $u[\![P]\!]^{\alpha}$ holds. Finally, $s[\![\text{ENABLED } A]\!]^{\alpha}$

holds iff $s[\![A]\!]^{\alpha}u$ holds for some state u. This implies that $\mathrm{WF}_v(A)$ and $\mathrm{SF}_v(A)$ represent weak and strong fairness assertions for the action A.

Definition 4.28 and the analogues of propositions 2.4 and 4.3 ensure that formulas of first-order TLA* are again invariant under stuttering equivalence. The proof of the following theorem is analogous to the proofs of theorems 2.5 and 4.5.

Theorem 4.29 *Assume that I is a first-order interpretation, σ and τ are behaviors, and α is a valuation of the rigid variables. For any formula F of first-order* TLA**, if $\sigma \simeq_{FFV(F)} \tau$ then $I,\sigma,\alpha \models F$ iff $I,\tau,\alpha \models F$.*

Concerning axiomatizations of first-order TLA*, we first remark that the axioms and rules of the original system Σ_{TLA^*} of section 2.3 remain sound. We add the following standard axioms and rules for reasoning about rigid quantification whose soundness is obvious.

$$(\exists\text{-I})_f \quad \vdash F[c/x] \rightarrow \exists x : F \qquad (\exists\text{-E})_f \quad \frac{\vdash F \rightarrow G}{\vdash (\exists x : F) \rightarrow G} \quad (x \notin \mathit{FRV}(G))$$

$$(\exists\text{-I})_p \quad \mathrel{|\!\sim} A[c/x] \rightarrow \exists x : A \qquad (\exists\text{-E})_p \quad \frac{\mathrel{|\!\sim} A \rightarrow B}{\mathrel{|\!\sim} (\exists x : A) \rightarrow B} \quad (x \notin \mathit{FRV}(B))$$

$$(\exists\text{-}\Diamond) \quad \vdash \Diamond\exists x : F \rightarrow \exists x : \Diamond F \qquad (\exists\text{-}\circ) \quad \mathrel{|\!\sim} \circ\exists x : F \rightarrow \exists x : \circ F$$

$$(\exists\text{-}\Diamond\langle\rangle) \quad \vdash \Diamond\langle\exists x : F\rangle_v \rightarrow \exists x : \Diamond\langle F\rangle_v$$

The "witness term" c in the introduction rules $(\exists\text{-I})_f$ and $(\exists\text{-I})_p$ must not contain any flexible variables. As in section 4.4, the converse implications of axioms $(\exists\text{-}\Diamond)$, $(\exists\text{-}\circ)$, and $(\exists\text{-}\Diamond\langle\rangle)$ are derivable. Note also that we can now state axiom $(\exists\text{-}\circ)$ since $\exists x : \circ F$ is a pre-formula. In particular, it can be used to derive the analogue of axiom $(\boldsymbol{\exists}\text{-}\Diamond\langle\circ\rangle)$.

Except for the introduction axiom $(\boldsymbol{\exists}\text{-P})$ for prophecy variables, the axioms and rules for flexible quantification introduced in section 4.4 remain sound with obvious modifications. In particular, the basic introduction axiom should now be stated in the form

$$(\boldsymbol{\exists}\text{-I}) \quad \vdash F[t/x] \rightarrow \boldsymbol{\exists} x : F$$

where F is any TLA* formula and t is a term. However, since terms do not contain temporal operators, an analogue of $(\boldsymbol{\exists}\text{-C})$ can no longer be derived. To recover the full strength of qpTLA*, one can either encode propositional logic in the first-order language or, preferably, add propositional variables directly.

Unfortunately, axiom ($\boldsymbol{\exists}$-P) is no longer sound for first-order TLA*: its soundness proof relied on König's lemma, which reqired the domain of quantification to be finite. One possible solution is to require that infinitely often the choice of the prophecy variable can be restricted to a single value; this idea is expressed in the following axiom (where I and N are as for ($\boldsymbol{\exists}$-P) and t is a finite set of terms):

$$(\boldsymbol{\exists}\text{-P})_1 \quad \vdash \begin{array}{l} \wedge\ \Box\exists x : I \\ \wedge\ \Box[\forall x' : I' \rightarrow \exists x : I \wedge N]_t \\ \wedge\ (\Box\Diamond\langle \mathbf{true}\rangle_t \rightarrow \Box\Diamond\langle \exists x : \forall x' : I' \rightarrow I \wedge N\rangle_t) \\ \rightarrow \boldsymbol{\exists}\, x : \Box I \wedge \Box[N \wedge t' \neq t]_{t,x} \end{array}$$

Theorem 4.30 *Axiom* $(\boldsymbol{\exists}\text{-}P)_1$ *is sound.*

Proof. Assume that I, α, and $\sigma = s_0 s_1 \ldots$ are such that

1. $\sigma[i\,..] \models \exists x : I$ for all $i \in \mathbb{N}$,
2. $\sigma[i\,..] \approx\!\models \forall x' : I' \rightarrow \exists x : I \wedge N$ whenever $s_{i+1}[\![t]\!] \neq s_i[\![t]\!]$, and

3a. either there exists $k \in \mathbb{N}$ such that $s_{i+1}[\![t]\!] = s_i[\![t]\!]$ for all $i \geq k$ or

3b. $\sigma[i\,..] \approx\!\models \exists x : \forall x' : I' \rightarrow I \wedge N \wedge t' \neq t$ holds for infinitely many $i \in \mathbb{N}$.

(We have suppressed I and α from the notation since they are fixed for the remainder of the proof.)

Writing s^d for the state that agrees with state s, except that x is assigned d, we construct a dag of pairs (i,d) where $i \in \mathbb{N}$ and $d \in |I|$ as follows:

- if the formula of condition (3b) holds of $\sigma[i\,..]$ then (i,d) appears in the dag if

$$s_i^d s_{i+1} s_{i+2} \ldots \approx\!\models \forall x' : I' \rightarrow I \wedge N$$

- otherwise, (i,d) appears in the dag if $s_i^d s_{i+1} s_{i+2} \ldots \models I$, and
- node (j,e) is a successor of node (i,d) iff $j = i+1$ and $d = e$ if $s_i[\![t]\!] = s_{i+1}[\![t]\!]$ and $s_i^d s_{i+1}^e s_{i+2} \ldots \approx\!\models N$ otherwise.

Observe that this definition and condition (1) imply that $s_i^d s_{i+1} s_{i+2} \models I$ whenever (i,d) is a node in the dag. Condition (1) above ensures that the dag contains some node (i,d) for every $i \in \mathbb{N}$, and condition (2) ensures that every node $(i+1,e)$ has some predecessor node (i,d). Inductively, it follows that the dag contains finite paths of arbitrary length.

In case condition (3a) holds then the definition of the dag ensures that every path of the form $(0,d_0)(1,d_1)\ldots(k,d_k)$ can be extended to an infinite path by repeating the value d_k. Otherwise, condition (3b) must hold, and therefore the dag is of width 1 at infinitely many levels. Therefore, (a generalized version of) König's lemma ensures the existence of an infinite path $\pi = (0,d_0)(1,d_1)\ldots(k,d_k)(k+1,d_{k+1})\ldots$.

In either case, we obtain an infinite path of the above form. Now define the behavior $\rho = r_0 r_1 \ldots$ where $r_i(x) \triangleq d_i$ and $r_i(y) \triangleq s_i(y)$ for all $y \not\equiv x$. Clearly, $\rho =_x \sigma$, and from the definition of the dag it is straightforward to prove that $r_{i+1}(x) \neq r_i(x)$ only if $r_{i+1}[\![t]\!] \neq r_i[\![t]\!]$. Finally, the syntactic conditions on I and N ensure that

$$\tau \models \Box I \wedge \Box[N \wedge t' \neq t]_{t,x}$$

which implies the assertion. Q.E.D.

Axiom $(\exists\text{-S})$ also needs some modification to be useful for first-order TLA*. First, the formulation shown in section 4.4 is not well-formed unless TLA* is extended by quantification over proposition variables as discussed above. More importantly, the axiom stated there introduces just a single stuttering step, and is therefore appropriate only if the amount of extra stuttering can be determined "statically" from the text of the formula, as was always the case in propositional TLA*. In the first-order case, a stronger version is necessary that allows the number of stuttering steps to be determined "dynamically", that is, from the current values of flexible variables. For the following we assume that the underlying first-order language contains the language of elementary arithmetic. We call an interpretation *arithmetical* if these symbols have the expected meaning. A suitable introduction axiom for stuttering variables can then be written as

$$\begin{array}{lll}(\exists\text{-S})_n & \vdash & (f \in \mathbb{N}) \wedge (\forall x : \Box[g \in \mathbb{N} \wedge g > 0]_x) \\ & & \rightarrow \exists x : \wedge\ x = f \\ & & \qquad\quad \wedge\ \Box[(x > 0 \wedge x' = x - 1 \wedge t' = t) \vee (x = 0 \wedge x' = g)]_{t,x} \\ & & \qquad\quad \wedge\ \Box\Diamond\langle \mathbf{true} \rangle_x\end{array}$$

where f is a term, g is a transition term, and t is a term (or finite set of terms) such that x does not appear in f, g or t. The left-hand side of $(\exists\text{-S})_n$ asserts type-correctness: f must initially evaluate to some natural number, and so must the transition term g, which should always be positive (the universal quantifier ensures that this is the case at every transition, not just when t changes its value). Under these assumptions, the axiom asserts that x can be chosen such that initially x equals f, that any step either decrements x without changing t or sets x to the value of g while possibly changing t, and that x changes its value infinitely often.

Theorem 4.31 *Axiom $(\exists\text{-}S)_n$ is sound for arithmetical interpretations.*

Proof. Assume that I is arithmetical and that α, and σ are such that the left-hand side of $(\exists\text{-S})_n$ holds, and let $\natural\sigma = r_0 r_1 \ldots$ be the stuttering-free variant of σ. Theorem 4.29 ensures that the left-hand side of $(\exists\text{-S})_n$ also holds of $\natural\sigma$. Now define finite sequences $\tau^0, \tau^1, \ldots$ of states (where $\tau^i = t^i_0 t^i_1 \ldots t^i_{n_i}$) as follows:

- t^0_0 is the x-variant of r_0 where $t^0_0(x) = n_0 = r_0[\![f]\!]$.

- t_0^{i+1} is the x-variant of r_{i+1} where $t_0^{i+1}(x) = n_{i+1} = r_i[\![g]\!]r_{i+1}$.
- For $j = 1, \ldots, n_i$, the state t_j^i is the x-variant of r_i where $t_j^i(x) = n_i - j$.

Define the behavior τ as the concatenation $\tau^0 \circ \tau^1 \circ \ldots$ of all τ^i. This definition ensures that $\tau \approx_x \natural\sigma$, hence also $\tau \approx_x \sigma$. Moreover, $t_0^0(x) = t_0^0[\![f]\!]$ (since x is assumed not to occur in f), and every transition either satisfies $x > 0 \land x' = x - 1 \land t' = t$ or $x = 0 \land x' = g$ (again, observe that x is assumed not to occur in g or t). Finally, $t_{n_i}^i(x) = 0$ for all $i \in \mathbb{N}$, whereas $t_0^{i+1}(x) > 0$ by assumption: $\Box[g \in \mathbb{N} \land g > 0]_x$ must be true of the behavior $\pi =_x \tau$ where π and τ agree in the valuation of x until state $t_{n_i}^i$, and π assigns x the value 1 thereafter. Therefore, x changes value infinitely often in τ, which completes the proof. Q.E.D.

Finally, it is often useful to be able to construct an "array" of auxiliary variables. For example, suppose we are specifying a multi-threaded server for use with multiple clients. The axioms that we have presented so far can be used to introduce auxiliary (history and prophecy) variables per client process during a refinement proof. During the refinement proof of the entire server specification we would then typically form an array (indexed by clients) that holds the value of the auxiliary variable appropriate for the client. For the statement of the following axiom we assume that the underlying first-order language contains arrays; we write $f[z]$ to denote the field of array f indexed by z. We say that an interpretation I *has arrays* if these symbols have the expected meaning.

$$(\exists\text{-F}) \quad (\forall z : \exists x : F \land \Box[t' \neq t]_x) \rightarrow \exists f : (\forall z : F[f[z]/x]) \land \Box[t' \neq t]_f$$

where t is a term (or finite set of terms) that does not contain x, and f is not free in F or t. Observe that ($\exists$-F) allows only non-stuttering auxiliary variables to be combined in an array. In fact, it would in general not be sound to combine stuttering variables: for all $k \in \mathbb{N}$, we could use $(\exists\text{-S})_n$ to introduce a stuttering variable s_k that forces any change of t to be separated by at least k stuttering steps. If we were allowed to form a product of all s_k, we would enforce infinite stuttering, and thus the resulting behavior would no longer be stuttering equivalent to the original one.

Theorem 4.32 *Axiom ($\exists$-F) is sound for interpretations that have arrays.*

Proof. Assume that I has arrays and that α and $\sigma = s_0 s_1 \ldots$ are such that the left-hand side of ($\exists$-F) holds; w.l.o.g., assume that σ is stuttering-free. We first show that for any $z \in |I|$ we may choose a behavior $\tau^z = t_0^z t_1^z \ldots$ such that $\tau^z =_x \sigma$ and $\tau^z \models F$: the assumption implies that for every z there exists some behavior $\rho^z \approx_x \sigma$ such that $\rho^z \models F \land \Box[t' \neq t]_x$. Because ρ is not allowed to change x while stuttering on t it follows that $\natural\rho =_x \sigma$. Moreover, theorem 4.29 ensures that $\natural\rho$ also satisfies $F \land \Box[t' \neq t]_x$, and can therefore be chosen as the behavior τ^z.

We now define the behavior $\tau = t_0 t_1 \ldots$ by letting $t_i(f)$, for all $i \in \mathbb{N}$, be the array such that $t_i(f)[z] = t_i^z(x)$, and $t_i(v) = s_i(v)$ for all $v \not\equiv f$. Clearly, $\tau =_f \sigma$, and $\tau \models F[f[z]/x]$ for all $z \in |I|$, which implies the assertion. Q.E.D.

This completes our presentation of axioms and rules for first-order TLA*. It is well known that first-order temporal logics are highly undecidable: without quantification over flexible variables, the validity problem is Π_1^1-complete [97, 96, 73]. Sistla [91] has shown that the addition of flexible quantification makes the problem Π_2^1-complete. We may therefore not hope for a traditional completeness result for first-order TLA*. Nevertheless, it can be shown [5] that the above rules are sufficient to establish completeness for restricted classes of specifications, similar to the semantic completeness proof given in [2].

4.8 Related Work

The limited expressiveness of unquantified propositional temporal logic, for example, with respect to S1S, the second-order theory of one successor, has been recognized as early as in Kamp's thesis [52] or in the seminal paper by Gabbay et al [39] who established the first completeness result for PTL. Several suggestions have been made to increase the expressiveness to the level of S1S, for example by grammar operators [104] or by fixed points [15]. Quantified propositional temporal logics have been studied beginning with the paper by Sistla et al [92] that established the close relationship between Büchi automata and temporal logic. Probably due to the predominant tradition of using temporal logic in a property-oriented style of specification, quantified temporal logics have not found wide use in actual temporal logic specifications. As we have argued in the introduction, the situation is completely different in the model-oriented style of TLA and TLA* where quantification is essential in order to avoid over-specification.

Beyond the realm of temporal logic, the problem of establishing refinement with the help of refinement mappings, extended by devices analogous to history and prophecy variables, has been extensively studied in semantic specification methods [2, 59, 67, 48]. A completeness result, without detailed proofs, for quantified PTL has been asserted by Kesten and Pnueli [54], and our proof follows their basic approach. The main differences are that their logic includes past-time as well as future-time operators and that it is sensitive to stuttering, which simplifies the semantics of quantification.

Chapter 5

Compositional Reasoning

5.1 The Problem

Modularity is a central concern in the design of specification methods. In general terms, modularity is the ability to reduce reasoning about a complete system to reasoning about its components. These components are not expected to operate in fully arbitrary environments. In the context of the complete system, each component can assume that its environment is to some extent well behaved, for instance that it adheres to certain communication protocols. Therefore, it is common to specify each component by describing both the function required of the component and the properties assumed of its environment. In the realm of sequential programs, for example, the requirements are postconditions and the assumptions are preconditions. In the realm of reactive systems, several forms of assumption-commitment specifications have been suggested [78, 47, 85, 3, 4, 95, 22, 105, 27, 49].

An assumption-commitment specification for a component of a reactive system consists of a formula A, which expresses assumptions about the environment, and a formula C, which expresses the requirements that an implementation of the component has to meet. Clearly, the meaning of such a specification depends somehow on A and C. In the simplest approach, the meaning is that if A holds then C holds, and if A does not hold then the implementation is completely unconstrained. One may therefore write the specification in the form $A \rightarrow C$, where $\rightarrow$ is the classical implication connective. This formulation seems intuitive; as we will see, however, stronger forms of assumption-commitment specifications are preferable for reasoning.

Suppose that we have two components specified by $A_1 \rightarrow C_1$ and $A_2 \rightarrow C_2$, and that we would like to prove that their composition satisfies the property P. Representing composition by conjunction and implementation by implication, we would have to prove that the formula $(A_1 \to C_1) \wedge (A_2 \to C_2) \to P$ is valid. In the composite system, each component is part of the other's environment; therefore, the assumption of each component may reflect the commitment of the other. So we would like to be able to use the commitment of each component to discharge the other's assumption. This argument suggests the following proof rule:

$$\frac{\begin{array}{c} C_1 \to A_2 \quad C_2 \to A_1 \\ C_1 \wedge C_2 \to P \end{array}}{(A_1 \to C_1) \wedge (A_2 \to C_2) \to P}$$

Unfortunately, this rule is unsound—it is an instance of circular reasoning.

One remedy is to strengthen the hypotheses in order to break the circularity, for example by demanding that $A_1 \vee A_2$ be valid; with this, the rule becomes classically sound but weak. Another remedy, which is more useful and common, is to strengthen assumption-commitment specifications. For each specification $A \to C$, one can add that C holds for longer than A in case A does not hold forever. The motivation for the stronger form of specification comes from the observation that no reasonable implementation of $A \to C$ will produce an execution that violates first C and later A. To do this, an implementation would have to predict the future behavior of the environment. In some formalisms, such prophetic implementations are excluded altogether, for example by continuity requirements.

Writing $A \overset{+}{\rhd} C$ for this stronger form of assumption-commitment specification, the next attempt to formulate a proof rule could be stated as

$$\frac{\begin{array}{c} A \wedge C_1 \wedge C_2 \to A_1 \wedge A_2 \\ A \wedge C_1 \wedge C_2 \to C \end{array}}{(A_1 \overset{+}{\rhd} C_1) \wedge (A_2 \overset{+}{\rhd} C_2) \to (A \overset{+}{\rhd} C)}$$

The first hypothesis serves to discharge the environment assumptions of the components. For example, when A is **true**, it expresses that the component guarantees C_1 and C_2 imply the component assumptions A_1 and A_2. The second hypothesis expresses that the final guarantee follows from the environment assumption A and the component guarantees C_1 and C_2. Unfortunately, this rule is still unsound. Nevertheless, we illustrate the use of the unsound rule, because reasoning with the unsound rule is similar to reasoning with its sound variants, and in order to justify later restrictions to the unsound rule. We consider two components X and Y that communicate through the variables x and y as shown in figure 5.1.

As a first example, we let

$$C_X \triangleq \Box[x' > x]_x \qquad \text{and} \qquad C_Y \triangleq \Box[y' > y]_y$$

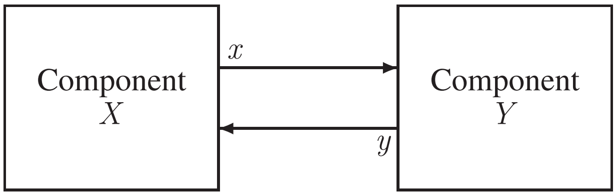


Figure 5.1: A simple example of composition.

and specify X by $C_Y \overset{+}{-\!\!\triangleright} C_X$ and X by $M_X \overset{+}{-\!\!\triangleright} M_Y$. That is, each component guarantees that the value of its output variable does not decrease, provided the value of its input variable does not decrease. According to the rule, we can derive $C_X \wedge C_Y$ from the conjunction of $C_X \overset{+}{-\!\!\triangleright} C_Y$ and $C_Y \overset{+}{-\!\!\triangleright} C_X$. That is, the composition of X and Y guarantees that the values of x and y do not decrease. This conclusion is correct.

As a second example, we let

$$C_X \triangleq \Diamond\Box odd(x) \qquad \text{and} \qquad C_Y \triangleq \Diamond\Box odd(y)$$

and specify X by $C_Y \overset{+}{-\!\!\triangleright} C_X$ and X by $M_X \overset{+}{-\!\!\triangleright} M_Y$. That is, each component guarantees that its output variable eventually becomes odd, provided its input variable eventually becomes odd. Again, the rule would allow us to deduce $C_X \wedge C_Y$. This conclusion is incorrect. To see this, consider the following specifications Φ_X and Φ_Y that imply $C_Y \overset{+}{-\!\!\triangleright} C_X$ and $C_X \overset{+}{-\!\!\triangleright} M_Y$, respectively:

$$\begin{aligned}
\Phi_X &\triangleq (x = 0) \wedge \Box[x' = y + 2]_x \wedge \mathrm{WF}_x(x' = y + 2) \\
\Phi_Y &\triangleq (y = 0) \wedge \Box[y' = x + 2]_y \wedge \mathrm{WF}_y(y' = x + 2)
\end{aligned}$$

Both x and y remain even throughout any behavior that satisfies $\Phi_X \wedge \Phi_Y$, so $\Phi_X \wedge \Phi_Y$ does not imply $C_X \wedge C_Y$. As this example illustrates, reasoning about assumption/guarantee specifications is more problematic for liveness properties than for safety properties.

The literature contains a number of sound variants of the unsound rule. Typically, the sound variants are justified using induction along computations. In one of the simplest cases, the reasoning may go: C_1 is not falsified before A_1, which is implied by C_2, so C_1 is not falsified before C_2; analogously, C_2 is not falsified before C_1; hence neither C_1 nor C_2 is ever falsified; therefore, P must hold, since $C_1 \wedge C_2$ implies P. There are many more delicate and more powerful arguments.

Despite the breadth of the literature, we believe that a few general logical ideas account for an interesting part of the work on composition. The purpose of this chapter is to present an abstract logic of specifications; the logic is independent of particular computational models, but it captures much of the reasoning common in formalisms with assumption-commitment specifications. Using the abstract logic, we derive proof rules for concrete specification methods. However, we do not attempt to capture every aspect of these specification methods; we focus on simple, basic results with broad applicability.

5.2 Abstract Principle

We justify the principle that underlies assumption-commitment style reasoning in a propositional intuitionistic logic. This logic is not meant as an actual object language used to formalize specifications, but it captures the abstract (and surprisingly weak) requirements that a logic of specifications should satisfy in order to allow for strong composition rules. Also note that although the logic we define is intuitionistic, we use classical principles, such as proofs by contradiction, at the meta level (for example, when reasoning about the semantics of the logic).

The atomic formulas of our logic should be thought of as representing entire specifications. The logic includes the standard connectives $\wedge$ and $\rightarrow$ of conjunction and implication. In addition, we define a connective $\xrightarrow{+}$, which will be useful to express assumption-commitment specifications. We now define the logic and its models more formally.

Assume given a nonempty set Σ and a pre-order $\sqsubseteq$ on Σ. We define $\sigma \sqsubset \tau$ to mean $\sigma \sqsubseteq \tau$ and $\tau \not\sqsubseteq \sigma$.[1] A set $S \subseteq \Sigma$ is downward closed if $\sigma \in S$ and $\tau \sqsubseteq \sigma$ imply that $\tau \in S$, for all $\sigma, \tau \in \Sigma$. We take Σ as the set of worlds of a Kripke frame [101, p.77], and define its accessibility relation as the inverse of $\sqsubseteq$ (so, τ is accessible from σ iff $\tau \sqsubseteq \sigma$). This accessibility relation is reflexive and transitive, and therefore such a Kripke frame, together with a valuation of atomic propositions that satisfies the usual stability condition, gives rise to a standard Kripke model of propositional intuitionistic logic, formally defined as follows:

Definition 5.1 (Semantics of the abstract logic)

1. *Atomic propositions $v \in \mathcal{V}$ are interpreted by a valuation $w : \Sigma \rightarrow 2^{\mathcal{V}}$; we say that v holds of σ or that v is true at σ iff $v \in w(\sigma)$, and we write $[\![v]\!]_w$ for the set $\{\sigma \in \Sigma : v \in w(\sigma)\}$ of worlds where v holds. It is required that every set $[\![v]\!]_w$ is downward closed. The pair (Σ, w) determines a Kripke model of intuitionistic logic.*

2. *The semantics of (complex) formulas is inductively given as the following relation $\models$ between worlds and formulas:*

[1] If $\sqsubseteq$ is not anti-symmetric, this definition of $\sigma \sqsubset \tau$ is strictly stronger than the simpler requrement $\sigma \sqsubseteq \tau$ and $\sigma \neq \tau$.

$$\begin{array}{lll} \sigma \models v & \textit{iff} & \sigma \in [\![v]\!]_w, \textit{for an atomic formula } v \in \mathcal{V} \\ \sigma \models F \wedge G & \textit{iff} & \sigma \models F \textit{ and } \sigma \models G \\ \sigma \models F \rightarrow G & \textit{iff} & \textit{for all } \tau \sqsubseteq \sigma\textit{: if } \tau \models F \textit{ then } \tau \models G \\ \sigma \models F \xrightarrow{+} G & \textit{iff} & \textit{for all } \tau \sqsubseteq \sigma\textit{: if } \rho \models F \textit{ for all } \rho \sqsubset \tau \textit{ then } \tau \models G \end{array}$$

The clauses in the above definition are standard, except for the semantics of the new connective $\xrightarrow{+}$. Intuitively, $F \rightarrow G$ asserts that G holds at least as long as F does, whereas $F \xrightarrow{+} G$ requires G to hold a little longer than F. It is a simple consequence of this definition that all formulas are true on downward closed sets.

Proposition 5.2 *For all formulas F, if $\sigma \models F$ then $\tau \models F$ for all $\tau \sqsubseteq \sigma$.*

In particular, $\tau \models F$ implies that $\rho \models F$ for all $\rho \sqsubset \tau$. It follows that the formula $F \xrightarrow{+} G$ is stronger than $F \rightarrow G$. Somewhat surprisingly, $\xrightarrow{+}$ can be defined from $\rightarrow$ for a large class of models, which are characterized by the following requirement.

Definition 5.3 (MF condition) *A Kripke model (Σ, w) is MF iff it satisfies the following condition for every atomic proposition $v \in \mathcal{V}$ and every $\sigma \in \Sigma$:*

If $\sigma \notin [\![v]\!]_w$ then there exists some $\tau \sqsubseteq \sigma$ such that $\tau \notin [\![v]\!]_w$ and $\rho \in [\![v]\!]_w$ holds for every $\rho \sqsubset \tau$.

The MF condition requires the existence of some smallest $\tau \sqsubseteq \sigma$ that does not satisfy v whenever σ does not satisfy v. This condition is certainly satisfied if $\sqsubset$ is well-founded, because every non-empty subset of a well-founded pre-order has a smallest element. However, the following example shows that the MF condition is strictly weaker than requiring well-foundedness of $\sqsubset$.

Example 5.4 (Real-time specifications) Assume given some set St of *states*, and let $\mathbb{R}^+$ denote the set of non-negative real numbers. A *trajectory* is a function $\sigma : I \rightarrow St$ from some closed initial interval $I = [0, x]$ of $\mathbb{R}^+$ to the set of states. If σ is a trajectory with domain $\mathrm{dom}(\sigma) = [0, x]$, we let $\mathrm{len}(\sigma) = x$ denote its length. We define a partial order $\sqsubseteq$ on the set Trj of trajectories by letting

$$\sigma \sqsubseteq \tau \quad \text{iff} \quad \mathrm{len}(\sigma) \leq \mathrm{len}(\tau) \text{ and } \sigma(r) = \tau(r) \text{ holds for all } r \leq \mathrm{len}(\sigma)$$

For any trajectory σ and $y \in \mathbb{R}^+$ such that $[0, y] \subseteq \mathrm{dom}(\sigma)$, we let $\sigma[..\, y]$ denote the trajectory τ of length y such that $\tau(r) = \sigma(r)$ for $0 \leq r \leq y$.

We can interpret our specification logic over trajectories. Intuitively, the atomic formulas of our logic represent specifications written in some underlying specification language. The valuation w will therefore be given by the semantics of the underlying language; we interpret $P \in w(\sigma)$ to mean that specification P has not been violated by time $\mathrm{len}(\sigma)$. We require the two following conditions to hold of this interpretation w of real-time specifications:

(1) $[\![P]\!]_w$ is downward closed, for any $P \in \mathcal{V}$.

(2) For all trajectories σ, if $P \in w(\sigma)$ then there exists some trajectory $\tau \sqsupset \sigma$ such that $P \in w(\tau)$.

The first condition is our usual requirement of downward closure; it is natural given the intended reading of $P \in w(\sigma)$ explained above. The second condition asserts, intuitively, that events that falsify a specification occur at specific time points. So, if P doesn't hold of σ then it will hold along a right-open initial prefix of σ. For example, suppose that P asserts that the component should react to some input within 10 time units. If the input occurs at time t, but the component does not react, then P will not have been violated by time $t+x$, for any $x < 10$, but it will have been violated from time $t+10$ onward. (Formally, we could extend our model to include trajectories with right-open domain $[0,x[$, or even with domain $[0,\infty[$, and define the interpretation w over such a trajectory σ by $P \in w(\sigma)$ iff $P \in w(\sigma[..y])$ for all $y < x$.)

We now show any model (Trj, w) that satisfies the two conditions above is MF. So suppose that $P \in \mathcal{V}$, that $\sigma \in Trj$ is some trajectory, and that $\sigma \notin [\![P]\!]_w$. Define the set

$$S \triangleq \{r \in \mathbb{R}^+ : \sigma[..r] \notin [\![P]\!]_w\}$$

Clearly, S is non-empty because $\sigma = \sigma[..\mathrm{len}(\sigma)]$, and therefore $\mathrm{len}(\sigma) \in S$. Let $s \triangleq \inf(S)$ denote the infimum of S in $\mathbb{R}^+$. Then condition (2) above ensures that $s \in S$, so we have $\sigma[..s] \notin [\![P]\!]w$. On the other hand, the choice of s as the infimum implies that $\tau \in [\![P]\!]w$ holds for all $\tau \sqsubset \sigma[..s]$. Q.E.D.

As with downward closure, the MF condition extends to arbitrary formulas.

Lemma 5.5 *If (Σ, w) is MF then for every formula F and all $\sigma \in \Sigma$, whenever $\sigma \not\models F$ then there exists some $\tau \sqsubseteq \sigma$ such that $\tau \not\models F$, but $\rho \models F$ holds for all $\rho \sqsubset \tau$.*

Proof. The proof is by induction over the structure of F.

$F \in \mathcal{V}$: by the MF condition.

$F \equiv G \wedge H$: Suppose that $\sigma \not\models G \wedge H$. Then $\sigma \not\models G$ or $\sigma \not\models H$, w.l.o.g. assume the former. By induction hypothesis, there is some $\tau \sqsubseteq \sigma$ such that $\tau \not\models G$, but $\rho \models G$ for all $\rho \sqsubset \tau$. If $\tau \models H$, then proposition 5.2 ensures $\rho \models H$ for all $\rho \sqsubset \tau$, and we are done. Otherwise, the induction hypothesis ensures the existence of some $\pi \sqsubseteq \tau$ such that $\pi \not\models H$, but $\rho \models H$ for all $\rho \sqsubset \pi$. Transitivity of $\sqsubseteq$ implies that $\rho \models G$ for all $\rho \sqsubset \pi$, and therefore we have $\pi \not\models G \wedge H$, but $\rho \models G \wedge H$ for all $\rho \sqsubset \pi$.

$F \equiv G \rightarrow H$: Suppose that $\sigma \not\models G \rightarrow H$, that is, for some $\pi \sqsubseteq \sigma$, we have $\pi \models G$ and $\pi \not\models H$. By induction hypothesis, there is $\tau \sqsubseteq \pi$ such that $\tau \not\models H$, but $\rho \models H$, and therefore $\rho \models G \rightarrow H$, for all $\rho \sqsubset \tau$. Downward closure implies that $\tau \models G$, and therefore $\tau \not\models G \rightarrow H$, which completes the proof.

$F \equiv G \overset{+}{\rightarrow} H$: Suppose that $\sigma \not\models G \overset{+}{\rightarrow} H$, that is, for some $\pi \sqsubseteq \sigma$, we have $\rho \models G$ for all $\rho \sqsubset \pi$, but $\rho \not\models H$. By the induction hypothesis, there is some $\tau \sqsubseteq \pi$ such that $\tau \not\models H$, but $\rho \models H$, and therefore $\rho \models G \overset{+}{\rightarrow} H$, holds for all $\rho \sqsubseteq \tau$. Transitivity of $\sqsubseteq$ implies that $\rho \models G$ holds for all $\rho \sqsubset \tau$, and hence $\tau \not\models G \overset{+}{\rightarrow} H$, which completes the proof. Q.E.D.

The following theorem relates the connectives $\rightarrow$ and $\overset{+}{\rightarrow}$; it is the crucial step for the remaining development of this chapter.

Theorem 5.6 (Definability of $\overset{+}{\rightarrow}$) *Assume that (Σ, w) satisfies the MF condition. For all $\sigma \in \Sigma$ and all formulas F and G,*

$$\sigma \models F \overset{+}{\rightarrow} G \qquad \textit{iff} \qquad \sigma \models (G \rightarrow F) \rightarrow G$$

Proof. "only if": Assume that $\sigma \models F \overset{+}{\rightarrow} G$, that $\tau \sqsubseteq \sigma$ and that $\tau \models G \rightarrow F$. We have to show that $\tau \models G$. Suppose not. By the assumption that (Σ, w) is MF and lemma 5.5, there exists some $\pi \sqsubseteq \tau$ such that $\pi \not\models G$, but $\rho \models G$ holds for all $\rho \sqsubset \pi$. Moreover, by downward closure (proposition 5.2), it follows that $\rho \models G \rightarrow F$, and therefore $\rho \models F$, holds for all $\rho \sqsubset \pi$. On the other hand, downward closure also implies that $\pi \models F \overset{+}{\rightarrow} G$. Hence, $\pi \rightarrow G$, and we have reached a contradiction.

"if": Assume now that $\sigma \models (G \rightarrow F) \rightarrow G$, that $\tau \sqsubseteq \sigma$, and that $\rho \models F$ for all $\rho \sqsubset \tau$. We have to show that $\tau \models G$. Suppose, to the contrary, that $\tau \not\models G$. Since $\tau \models (G \rightarrow F) \rightarrow G$ holds by downward closure, a contradiction can be obtained by proving $\tau \models G \rightarrow F$. Assume that $\rho \sqsubseteq \tau$ and that $\rho \models G$. Either $\tau \sqsubseteq \rho$ or $\rho \sqsubset \tau$. If $\tau \sqsubseteq \rho$, downward closure gives us $\tau \models G$ in contradiction with our assumption. Hence, $\rho \sqsubset \tau$, and therefore $\rho \models F$ by assumption. It follows that $\tau \models G \rightarrow F$, which completes the proof. Q.E.D.

From now on, we assume that our models satisfy the MF condition, and formally treat $F \overset{+}{\rightarrow} G$ as a shorthand notation for $(G \rightarrow F) \rightarrow G$. In this way, we can reason syntactically about $\rightarrow$ and $\overset{+}{\rightarrow}$, using any standard axiomatization of propositional intuitionistic logic. The following lemma lists a few simple results; the notation $F_1, \ldots, F_n \vdash F$ means that F can be derived from assumptions $F_1, \ldots, F_n$ or, equivalently, that the conjunction of $F_1, \ldots, F_n$ implies F.

Lemma 5.7 *The following sequents are derivable:*

$$F \xrightarrow{+} G, F \xrightarrow{+} (G \rightarrow H) \;\vdash\; F \xrightarrow{+} H \tag{5.1}$$

$$F_1 \xrightarrow{+} G_1, \ldots, F_n \xrightarrow{+} G_n \;\vdash\; \bigwedge_{i=1}^{n} F_i \xrightarrow{+} \bigwedge_{i=1}^{n} G_i \tag{5.2}$$

$$F \rightarrow G, G \xrightarrow{+} H \;\vdash\; F \xrightarrow{+} H \tag{5.3}$$

$$F \xrightarrow{+} G, G \rightarrow H \;\vdash\; F \xrightarrow{+} H \tag{5.4}$$

$$F \xrightarrow{+} G \;\vdash\; F \rightarrow G \tag{5.5}$$

$$F \xrightarrow{+} F \;\vdash\; F \tag{5.6}$$

Proof. We give derivations for assertions (5.2) and (5.6); the other facts are shown similarly.

By the deduction theorem for propositional intuitionistic logic, in order to show (5.2) for $n = 2$ (the generalization to arbitrary n is obvious), it suffices to derive

$$F_1 \xrightarrow{+} G_1, F_2 \xrightarrow{+} G_2, G_1 \wedge G_2 \rightarrow F_1 \wedge F_2 \vdash \; G_1 \wedge G_2$$

(1)	$(G_1 \rightarrow F_1) \rightarrow G_1$	(assumption)
(2)	$(G_2 \rightarrow F_2) \rightarrow G_2$	(assumption)
(3)	$G_1 \wedge G_2 \rightarrow F_1$	(assumption)
(4)	$G_1 \wedge G_2 \rightarrow F_2$	(assumption)
(5)	$G_2 \rightarrow (G_1 \rightarrow F_1)$	(3)
(6)	$G_2 \rightarrow G_1$	(5)(1)
(7)	$G_2 \rightarrow F_2$	(6)(4)
(8)	G_2	(7)(2)
(9)	G_1	(8)(6)
(10)	$G_1 \wedge G_2$	(9)(8)

The derivation of (5.6) is even simpler:

(1)	$(F \rightarrow F) \rightarrow F$	(assumption)
(2)	$F \rightarrow F$	(tautology)
(3)	F	(1)(2)

Q.E.D.

Sequents (5.1) to (5.4) state implication-like properties of $\xrightarrow{+}$. Sequent (5.5) asserts that $\xrightarrow{+}$ is stronger than $\rightarrow$, as we have already observed from the semantics of the operators. Remembering the semantics of $\xrightarrow{+}$, sequent (5.6) can be understood as an abstract formulation of computational induction.

In our applications of the abstract logic to prove composition rules for assumption-commitment specifications, Σ will (roughly) represent a set of computations, and

$\sigma \sqsubseteq \tau$ holds if computation σ may evolve to computation τ. Atomic propositions of the abstract logic correspond to specifications, so $\sigma \models F$ means that computation σ is allowed by specification F. Consequently, $\sigma \models F \rightarrow G$ means that G holds for at least as long as F along computation σ, whereas $\sigma \models F \stackrel{+}{\rightarrow} G$ means that G holds for strictly longer than F along σ. Either of the formulas $F \rightarrow G$ or $F \stackrel{+}{\rightarrow} G$ can represent assumption-commitment specifications; in fact, both interpretations occur in the literature.

On the other hand, we interpret the formula $F \rightarrow G$ as asserting that F refines G, because $F \rightarrow G$ is valid iff every computation that is allowed by F is also allowed by G. Refinement of assumption-commitment specifications is thus expressed by either $(F_1 \rightarrow G_1) \rightarrow (F \rightarrow G)$ or by $(F_1 \stackrel{+}{\rightarrow} G_1) \rightarrow (F \stackrel{+}{\rightarrow} G)$. Our next theorem establishes special cases of such formulas: they allow to replace the original assumption G by a weaker assumption F, using the commitment H. In this respect, they contain the essence of strong composition rules for assumption-commitment specifications. Despite their apparent circularity, the rules are sound because of the distinction between the connectives $\rightarrow$ and $\stackrel{+}{\rightarrow}$.

Theorem 5.8 *The following sequents are derivable:*

$$G \rightarrow (H \rightarrow F) \ \vdash \ (F \stackrel{+}{\rightarrow} H) \rightarrow (G \stackrel{+}{\rightarrow} H) \tag{5.7}$$

$$G \rightarrow (F \wedge H \stackrel{+}{\rightarrow} F) \ \vdash \ (F \rightarrow H) \rightarrow (G \rightarrow H) \tag{5.8}$$

Proof. We start by proving (5.7). As a first step, we derive

$$G \rightarrow (H \rightarrow F), F \stackrel{+}{\rightarrow} H \vdash G \rightarrow (H \stackrel{+}{\rightarrow} H)$$

(1)	$G \rightarrow (H \rightarrow F)$	(assumption)
(2)	$F \stackrel{+}{\rightarrow} H$	(assumption)
(3)	G	(assumption)
(4)	$H \rightarrow F$	(1)(3)
(5)	$H \stackrel{+}{\rightarrow} H$	(4)(2)(5.3)

From here, we continue as follows:

(1)	$G \rightarrow (H \rightarrow F)$	(assumption)
(2)	$F \stackrel{+}{\rightarrow} H$	(assumption)
(3)	$G \rightarrow (H \stackrel{+}{\rightarrow} H)$	(see above)
(4)	$(H \stackrel{+}{\rightarrow} H) \rightarrow H$	(5.6)
(5)	$G \rightarrow H$	(3)(4)
(6)	$G \rightarrow F$	(5)(1)
(7)	$G \stackrel{+}{\rightarrow} H$	(6)(2)(5.3)

The derivation of (5.8) also relies essentially on the “induction principle” (5.6):

(1)	$G \to (F \wedge H \xrightarrow{+} F)$	(assumption)
(2)	$F \to H$	(assumption)
(3)	G	(assumption)
(4)	$F \wedge H \xrightarrow{+} F$	(3)(1)
(5)	$F \to F \wedge H$	(2)
(6)	$F \wedge H \xrightarrow{+} F \wedge H$	(4)(5)(5.4)
(7)	$F \wedge H$	(6)(5.6)
(8)	H	(7)

Q.E.D.

As a corollary to theorem 5.8, we obtain the following rules for proving that a conjunction of “low-level” assumption-commitment specifications (in either style) refines a “high-level” specification.

Corollary 5.9 *Let $n \in \mathbb{N}$ and assume that F, G, F_i, and G_i (for $i = 1, \dots, n$) are formulas. Then:*

$$\begin{pmatrix} F \to \Big(\bigwedge_{i=1}^{n} G_i \to \bigwedge_{i=1}^{n} F_i\Big) \\ \wedge \\ F \xrightarrow{+} \Big(\bigwedge_{i=1}^{n} G_i \to G\Big) \end{pmatrix} \to \Big(\bigwedge_{i=1}^{n} (F_i \xrightarrow{+} G_i) \to (F \xrightarrow{+} G)\Big) \qquad (5.9)$$

$$\begin{pmatrix} F \to \Big(\bigwedge_{i=1}^{n} F_i \wedge G_i \xrightarrow{+} \bigwedge_{i=1}^{n} F_i\Big) \\ \wedge \\ F \xrightarrow{+} \Big(\bigwedge_{i=1}^{n} G_i \to G\Big) \end{pmatrix} \to \Big(\bigwedge_{i=1}^{n} (F_i \to G_i) \to (F \to G)\Big) \qquad (5.10)$$

Proof. By intuitionistic logic, we may identify the asserted formulas with sequents whose hypotheses are the two conjuncts to the left of the main implication connective. From the first hypothesis of (5.9) and (5.7) we obtain

$$\Big(\bigwedge_{i=1}^{n} F_i \xrightarrow{+} \bigwedge_{i=1}^{n} G_i\Big) \to \Big(F \xrightarrow{+} \bigwedge_{i=1}^{n} G_i\Big)$$

Using (5.2), this implies

$$\bigwedge_{i=1}^{n} (F_i \xrightarrow{+} G_i) \to \Big(F \xrightarrow{+} \bigwedge_{i=1}^{n} G_i\Big)$$

and (5.9) follows by using the second hypothesis, (5.1), and simple intuitionistic reasoning.

Similarly, from the first hypothesis of (5.10) and (5.8) we deduce

$$\Big(\bigwedge_{i=1}^{n} F_i \to \bigwedge_{i=1}^{n} G_i\Big) \to \Big(F \xrightarrow{+} \bigwedge_{i=1}^{n} G_i\Big)$$

and by the distributivity of conjunction over implication this implies

$$\bigwedge_{i=1}^{n} (F_i \to G_i) \to \Big(F \to \bigwedge_{i=1}^{n} G_i\Big)$$

Assertion (5.10) follows by the second hypothesis and simple intuitionistic logic. Q.E.D.

The composition rules (5.9) and (5.10) resemble the unsound rule discussed in section 5.1. They avoid the problem of circularity because of the distinction between $\to$ and $\xrightarrow{+}$. Either rule can be chosen as the basis for a composition rule for assumption-commitment specifications in a concrete formalism. In practice, the first rule (5.9) is somewhat easier to use, because its main (first) premise does not contain $\xrightarrow{+}$, and its occurrence in the second premise is less "critical": often, one will be able to prove $\bigwedge G_i \to G$ directly. However, the associated format $F \xrightarrow{+} G$ of assumption-commitment specifications is more demanding of the implementation because it requires the commitment to be maintained (at least for a while) even after the assumption has been violated.

Remark. We have derived the "abstract composition rules" (5.9) and (5.10) in a propositional logic. Nevertheless, the rules and their proof can be generalized beyond finite conjunctions, because universal quantification commutes with both $\to$ and $\xrightarrow{+}$. It is, however, interesting to observe that

$$\forall x(F \xrightarrow{+} G) \to (\forall x F \xrightarrow{+} \forall x G)$$

is semantically justified in our models of interest, although it is not generally true (and is therefore not provable) in intuitionistic predicate logic.

5.3 Application to TLA*

The abstract principles from section 5.2 can be used quite directly to motivate a format of assumption-commitment specifications and associated proof rules for (stuttering-invariant) linear-time temporal logics. We mainly follow Abadi and Lamport [4] who have studied the problem in the context of TLA, and reestablish their main results. We believe that the exposition of the principles of reasoning is simplified by use of the tools of section 5.2.

5.3.1 A composition rule for TLA*

In chapters 2 and 4 we have defined the semantics $\sigma \models F$ of TLA* formulas over infinite sequences of states. In the following, it will be convenient to extend this relation to finite sequences of states: a formula F holds of a finite sequence ρ (written $\rho \models F$) if ρ is empty or if there exists an infinite sequence σ that extends ρ such that F holds of σ. Following the standard definition [12], a formula F expresses a safety property if F holds of an infinite sequence σ whenever it holds of all finite prefixes of σ.

We temporarily extend TLA* by the following additional connectives that are useful to express assumption-commitment specifications. We will see in section 5.3.2 below that these connectives can actually be defined in TLA*.

- $\sigma \models \mathcal{C}(F)$ iff F holds of all finite prefixes of σ.
- $\sigma \models F \mathrel{-\!\triangleright} G$ iff, for all (finite or infinite) prefixes ρ of σ, if F holds of ρ, then so does G.
- $\sigma \models F \overset{+}{\mathrel{-\!\triangleright}} G$ iff both:
  1. for all $n \geq 0$, if F holds of $\sigma[..n]$, then G holds of $\sigma[..n+1]$, and
  2. if F holds of σ then so does G.

In particular, $\mathcal{C}(F)$ denotes the strongest safety property implied by F, and F is a safety property iff F and $\mathcal{C}(F)$ are equivalent. Observe that for any finite sequence ρ, a formula F holds of ρ iff $\mathcal{C}(F)$ holds of ρ.

Lemma 5.10 *Assume that F and G are* TLA* *formulas.*

1. *If G is a safety property then so are $F \mathrel{-\!\triangleright} G$ and $F \overset{+}{\mathrel{-\!\triangleright}} G$.*
2. $(F \mathrel{-\!\triangleright} G) \leftrightarrow (\mathcal{C}(F) \mathrel{-\!\triangleright} \mathcal{C}(G)) \wedge (F \Rightarrow G)$
3. $(F \overset{+}{\mathrel{-\!\triangleright}} G) \leftrightarrow (\mathcal{C}(F) \overset{+}{\mathrel{-\!\triangleright}} \mathcal{C}(G)) \wedge (F \Rightarrow G)$
4. $\mathcal{C}(F \mathrel{-\!\triangleright} G) \leftrightarrow \mathcal{C}(F) \mathrel{-\!\triangleright} \mathcal{C}(G)$
5. $\mathcal{C}(F \overset{+}{\mathrel{-\!\triangleright}} G) \leftrightarrow \mathcal{C}(F) \overset{+}{\mathrel{-\!\triangleright}} \mathcal{C}(G)$
6. *$F \mathrel{-\!\triangleright} G$ is valid iff $F \Rightarrow G$ is valid.*

Proof.

1. Assume that G is a safety property and that $\sigma[..n] \models F \mathrel{-\!\!\triangleright} G$ for all $n \geq 0$; we have to show that $\sigma \models F \mathrel{-\!\!\triangleright} G$. Let ρ be some prefix of σ and suppose that $\rho \models F$. If ρ is finite, that is, $\rho = \sigma[..n]$ for some n, then $\rho \models F \mathrel{-\!\!\triangleright} G$ by assumption, and therefore also $\rho \models G$, by the definition of $\mathrel{-\!\!\triangleright}$. If ρ is infinite, then $\rho = \sigma$, and the assumption $\rho \models F$ implies that $\sigma[..n] \models F$ holds for all $n \geq 0$. Therefore, it follows that $\sigma[..n] \models G$ holds for all $n \geq 0$, since $\sigma[..n] \models F \mathrel{-\!\!\triangleright} G$ by assumption. But since G is a safety property, it follows that $\sigma \models G$.

   The proof for $\overset{+}{\mathrel{-\!\!\triangleright}}$ is analogous.

2. "$\Rightarrow$": Assume that $\sigma \models F \mathrel{-\!\!\triangleright} G$. If $\sigma \models F$ then $\sigma \models G$ holds by definition, therefore we have $\sigma \models F \Rightarrow G$. It remains to prove $\sigma \models \mathcal{C}(F) \mathrel{-\!\!\triangleright} \mathcal{C}(G)$. By asertion 1, this is a safety property, so it suffices to prove $\sigma[..n] \models \mathcal{C}(F) \mathrel{-\!\!\triangleright} \mathcal{C}(G)$, for all $n \geq 0$. Therefore, let τ be some prefix of $\sigma[..n]$, that is, $\tau = \sigma[..m]$ for some $m \leq n$, and assume that $\sigma[..m] \models \mathcal{C}(F)$. The assumption $\sigma \models F \mathrel{-\!\!\triangleright} G$ implies $\sigma[..m] \models G$, which proves the assertion.

   "$\Leftarrow$": immediate from the definition of $\sigma \models F \mathrel{-\!\!\triangleright} G$.

3. Similar to the proof of assertion 2.

4. "$\Rightarrow$": Assume that $\sigma \models \mathcal{C}(F \mathrel{-\!\!\triangleright} G)$, that τ is some prefix of σ, and that $\tau \models \mathcal{C}(F)$. We have to prove that $\tau \models \mathcal{C}(G)$. So assume that $n \geq 0$. From the assumptions, we conclude $\tau[..n] \models F$ and $\tau[..n] \models F \mathrel{-\!\!\triangleright} G$, hence it follows that $\tau[..n] \models G$, as desired.

   "$\Leftarrow$": Assume that $\sigma \models \mathcal{C}(F) \mathrel{-\!\!\triangleright} \mathcal{C}(G)$ and that $n \geq 0$. We have to show that $\sigma[..n] \models F \mathrel{-\!\!\triangleright} G$. So assume further that $m \leq n$ and that $\sigma[..m] \models F$. Therefore, $\sigma[..m] \models \mathcal{C}(F)$, and the assumption implies $\sigma[..m] \models \mathcal{C}(G)$, which means $\sigma[..m] \models G$.

5. Similar to the proof of assertion 4.

6. By assertion 2, $(F \mathrel{-\!\!\triangleright} G) \Rightarrow (F \Rightarrow G)$, and therefore the validity of $F \mathrel{-\!\!\triangleright} G$ implies that of $F \Rightarrow G$.

   Conversely, assume that $F \Rightarrow G$ is valid, let σ be any sequence of states, and assume that $\rho \models F$ for some prefix ρ of σ. If ρ is empty, then $\rho \models G$ holds trivially. Otherwise, choose some infinite state sequence τ such that ρ is a prefix of τ and $\tau \models F$. By assumption, it follows that $\tau \models G$, and therefore $\rho \models G$, since ρ is a prefix of τ. Hence, $\sigma \models F \mathrel{-\!\!\triangleright} G$ holds for any σ. Q.E.D.

For a finite and non-empty sequence $\rho = s_0 s_1 \ldots s_n$, let $\uparrow\rho$ denote the infinite sequence that is obtained by repeating the last state of ρ indefinitely, that is, define $\uparrow\rho = s_0 s_1 \ldots s_n s_n s_n \ldots$.

Lemma 5.11 *Assume that* ρ *is a finite and non-empty sequence of states. Then* $\rho \models F$ *iff* $\uparrow\rho \models \mathcal{C}(F)$.

Proof. The "if" part is immediate. For the "only if" part, assume that $\rho \models F$ where $\rho = \langle s_0, s_1, \ldots, s_n \rangle$. Hence there exists some infinite sequence of states τ such that $\rho \circ \tau \models F$. Let ρ^k denote the finite sequence of states that is obtained from ρ by appending k copies of s_n. Because F is invariant under stuttering by theorem 4.29, it follows that $\rho^k \circ \tau \models F$. Therefore $\pi \models F$ holds for any prefix π of $\uparrow\rho$, which implies that $\uparrow\rho \models \mathcal{C}(F)$. Q.E.D.

Lemma 5.11 corresponds well to the intuitive interpretation of safety properties as asserting that "nothing bad ever happens": at any point during the behavior, if the system terminated now (represented by infinite stuttering), it would not have violated the safety part of the specification. It also simplifies compositional reasoning, as we will see in the following lemma.

The interpretation of temporal logic formulas over finite sequences of states gives rise to a model of our abstract logic from section 5.2, which considers TLA* formulas as atomic propositions. Specifically, let $(\Sigma, \sqsubseteq)$ be the set of finite state sequences with the prefix order, and define a valuation w by $F \in w(\rho)$ iff $\rho \models F$; this valuation clearly satisfies the requirement of downward closure. Hence, definition 5.1 gives rise to a semantics $\overset{c}{\models}$ of the abstract logic. The connectives of the abstract logic correspond closely to the temporal logic operators introduced above:

Lemma 5.12 *Assume that F, G are* TLA* *formulas, and that $\rho \in \Sigma$.*

1. $\rho \overset{c}{\models} F \rightarrow G$ *iff* $\rho \models F \mathrel{-\!\triangleright} G$.
2. $\rho \overset{c}{\models} F \overset{+}{\rightarrow} G$ *iff* $\rho \models F \overset{+}{\mathrel{-\!\triangleright}} G$.
3. $\rho \overset{c}{\models} F \wedge G$ *iff* $\rho \models \mathcal{C}(F) \wedge \mathcal{C}(G)$.

Proof. We only give the proof for assertion 3; the proof of the other assertions is straightforward. Assume that $\rho \overset{c}{\models} F \wedge G$, hence $\rho \overset{c}{\models} F$ and $\rho \overset{c}{\models} G$. By definition, this implies $\rho \models F$ and $\rho \models G$. If ρ is empty, the conclusion $\rho \models \mathcal{C}(F) \wedge \mathcal{C}(G)$ holds trivially; otherwise, lemma 5.11 ensures that $\uparrow\rho \models \mathcal{C}(F)$ and $\uparrow\rho \models \mathcal{C}(G)$, which implies $\rho \models \mathcal{C}(F) \wedge \mathcal{C}(G)$. Conversely, if $\rho \models \mathcal{C}(F) \wedge \mathcal{C}(G)$, then $\rho \models \mathcal{C}(F)$ and $\rho \models \mathcal{C}(G)$, which immediately implies $\rho \overset{c}{\models} F$ and $\rho \overset{c}{\models} G$. Q.E.D.

In particular, lemma 5.12 implies that if F and G are safety properties then the "abstract" and "temporal" connectives correspond exactly, and we can use this correspondence to justify a composition rule for safety properties from theorem 5.9.

Theorem 5.13 *Assume that A, C, A_i, and C_i (for $i = 1, \ldots, n$) are* TLA* *formu-*

las that express safety properties. Then we have the following rule:

$$
\frac{\begin{array}{l} A \wedge \bigwedge_{i=1}^{n} C_i \Rightarrow \bigwedge_{i=1}^{n} A_i \\ A \stackrel{+}{-\!\!\triangleright} \Big(\bigwedge_{i=1}^{n} C_i -\!\!\triangleright C \Big) \end{array}}{\bigwedge_{i=1}^{n} (A_i \stackrel{+}{-\!\!\triangleright} C_i) \Rightarrow (A \stackrel{+}{-\!\!\triangleright} C)}
$$

Proof. Assume the hypotheses of the rule. If σ is an infinite state sequence such that $\sigma \models \bigwedge_{i=1}^{n} (A_i \stackrel{+}{-\!\!\triangleright} C_i)$ holds, then $\rho \models \bigwedge_{i=1}^{n} (A_i \stackrel{+}{-\!\!\triangleright} C_i)$ holds for every finite prefix ρ of σ, and therefore $\rho \models A \stackrel{+}{-\!\!\triangleright} C$ follows for every finite prefix by lemma 5.12 and theorem 5.9. But, lemma 5.10.1 ensures that $A \stackrel{+}{-\!\!\triangleright} C$ is again a safety property, which implies the conclusion. Q.E.D.

The composition rule of theorem 5.13 can be extended to arbitrary TLA* formulas, using the equivalences from lemma 5.10. We only need to add closure operators as appropriate, and add a third hypothesis for the liveness part.

Theorem 5.14 *Assume that A, C, A_i, and C_i (for $i = 1, \ldots, n$) are* TLA* *formulas. Then we have the following rule:*

$$
\frac{\begin{array}{l} \mathcal{C}(A) \wedge \bigwedge_{i=1}^{n} \mathcal{C}(C_i) \Rightarrow \bigwedge_{i=1}^{n} A_i \\ \mathcal{C}(A) \stackrel{+}{-\!\!\triangleright} \Big(\bigwedge_{i=1}^{n} \mathcal{C}(C_i) -\!\!\triangleright \mathcal{C}(C) \Big) \\ A \wedge \bigwedge_{i=1}^{n} C_i \Rightarrow C \end{array}}{\bigwedge_{i=1}^{n} (A_i \stackrel{+}{-\!\!\triangleright} C_i) \Rightarrow (A \stackrel{+}{-\!\!\triangleright} C)}
$$

Proof. The first two hypotheses and theorem 5.13 imply

$$
\bigwedge_{i=1}^{n} (\mathcal{C}(A_i) \stackrel{+}{-\!\!\triangleright} \mathcal{C}(C_i)) \Rightarrow (\mathcal{C}(A) \stackrel{+}{-\!\!\triangleright} \mathcal{C}(C))
$$

since $A_i \Rightarrow \mathcal{C}(A_i)$ is valid. Using lemma 5.10.3, it remains to prove

$$
\bigwedge_{i=1}^{n} (A_i \stackrel{+}{-\!\!\triangleright} C_i) \Rightarrow (A \Rightarrow C)
$$

First, we use again theorem 5.13, substituting $\bigwedge \mathcal{C}(C_i)$ for C, to establish

$$\bigwedge_{i=1}^{n} (\mathcal{C}(A_i) \stackrel{+}{\rightarrow\!\triangleright} \mathcal{C}(C_i)) \Rightarrow (\mathcal{C}(A) \stackrel{+}{\rightarrow\!\triangleright} \bigwedge_{i=1}^{n} \mathcal{C}(C_i))$$

By lemma 5.10.3 we conclude

$$\bigwedge_{i=1}^{n} (\mathcal{C}(A_i) \stackrel{+}{\rightarrow\!\triangleright} \mathcal{C}(C_i)) \Rightarrow (\mathcal{C}(A) \Rightarrow \bigwedge_{i=1}^{n} \mathcal{C}(C_i))$$

The first hypothesis of the rule implies

$$\bigwedge_{i=1}^{n} (\mathcal{C}(A_i) \stackrel{+}{\rightarrow\!\triangleright} \mathcal{C}(C_i)) \Rightarrow (\mathcal{C}(A) \Rightarrow \bigwedge_{i=1}^{n} A_i)$$

Using again lemma 5.10.3 and the validity of $A \Rightarrow \mathcal{C}(A)$, we obtain

$$\bigwedge_{i=1}^{n} (A_i \stackrel{+}{\rightarrow\!\triangleright} C_i) \Rightarrow (A \Rightarrow \bigwedge_{i=1}^{n} A_i)$$

and yet another application of lemma 5.10.3 (to the left-hand side of the above implication) yields

$$\bigwedge_{i=1}^{n} (A_i \stackrel{+}{\rightarrow\!\triangleright} C_i) \Rightarrow (A \Rightarrow \bigwedge_{i=1}^{n} C_i)$$

and the desired conclusion follows by the third hypothesis of the rule. Q.E.D.

To apply this composition rule, the low-level assumptions A_i have to be deduced from a conjunction of safety properties. Hence, in practice, they commonly are safety properties themselves; this restriction to safety properties is consistent with the informal discussion and the examples of section 5.1. When A_i is not a safety property, we can use the equivalence

$$A_i \stackrel{+}{\rightarrow\!\triangleright} C_i \quad \leftrightarrow \quad (\mathcal{C}(A_i) \stackrel{+}{\rightarrow\!\triangleright} \mathcal{C}(C_i)) \wedge (A_i \rightarrow C_i)$$

of lemma 5.10 to apply the composition rule to the safety part of the specification, and use ordinary temporal reasoning for the liveness part.

The first and third hypotheses of the composition rule can be established using standard TLA* proof rules, and some auxiliary rules to deal with closures. A strategy for reducing the proof of the second hypothesis to the proof of ordinary implications is discussed in [4]. An alternative strategy consists in using the following rule:

$$\frac{\begin{array}{c} F \wedge G \rightarrow \Box P \\ \Box[P]_v \wedge G \rightarrow H \end{array}}{F \stackrel{+}{\rightarrow\!\triangleright} (G \rightarrow\!\triangleright H)}$$

where P is a state predicate, v a term (or finite set of terms), and F, G, and H arbitrary TLA* formulas. This rule enables us to derive a formula with $\rightarrowtriangle$ and $\overset{+}{\rightarrowtriangle}$ from two classical implications. The state predicate P plays the role of an invariant, guaranteed by F and G according to the first hypothesis. The second hypothesis says, roughly, that H follows from G together with the invariant.

Thus, the techniques for verifying the hypotheses of the composition rule are rather specific to TLA*. On the other hand, the composition rule is general, and follows fairly directly from the algebraic arguments of section 5.2.

5.3.2 Syntactic definitions

In section 5.3 we have introduced operators $\mathcal{C}$, $\rightarrowtriangle$, and $\overset{+}{\rightarrowtriangle}$ as additional connectives and have defined their semantics. We now show that these connectives can actually be defined as derived connectives in TLA*.[2]

We first consider the definition of $\mathcal{C}(F)$, for a TLA* formula F. Assume that $x_1, \ldots, x_n$ are all free flexible variables in F, let $t, y_1, \ldots, y_n$ be flexible variables different from $x_1, \ldots, x_n$, and let u be a rigid variable that does not occur in F. Then:

$$\mathcal{C}(F) \quad \leftrightarrow \quad \boldsymbol{\forall}\, t : \forall u : \begin{pmatrix} \Box[t' \neq u]_t \wedge \mathrm{WF}_t(t' \neq u) \rightarrow \\ \boldsymbol{\exists}\, y_1, \ldots, y_n : \wedge\, \Box\Big(t = u \rightarrow \bigwedge_{i=1}^{n} x_i = y_i\Big) \\ \wedge\, F[y_1/x_1, \ldots, y_n/x_n] \end{pmatrix}$$

Roughly, the antecedent of the implication, $\Box[t' \neq u]_t \wedge \mathrm{WF} t t' \neq u$, says that z may equal u for some time and then becomes different from u, say at time n. The consequent,

$$\boldsymbol{\exists}\, y_1, \ldots, y_n : \Box\Big(t = u \rightarrow \bigwedge_{i=1}^{n} x_i = y_i\Big) \wedge F[y_1/x_1, \ldots, y_n/x_n]$$

requires that the sequence of values for $x_1, \ldots, x_n$ up to time $n-1$ can be extended to an infinite sequence of values for $y_1, \ldots, y_n$ that satisfies F. The quantifiers $\boldsymbol{\forall}\, t$ and $\forall u$ are in effect quantifying over all n.

The technique used in the definition of $\mathcal{C}$ yields a definition of $\rightarrowtriangle$ for safety properties. Under similar assumptions as above and writing $\vec{x}$ and $\vec{y}$ for $x_1, \ldots, x_n$

[2]The operators could be defined for TLA in the same way as for TLA*.

and $y_1, \dots, y_n$ we have:

$$\mathcal{C}(F) \mathrel{-\!\!\triangleright} \mathcal{C}(G) \;\; \leftrightarrow \;\; \boldsymbol{\forall}\, t : \forall u : \begin{pmatrix} \Box[t' \neq u]_t \wedge \mathrm{WF}_t(t' \neq u) \\ \rightarrow \\ \begin{pmatrix} \boldsymbol{\exists}\, \vec{y} : \Box(u = z \rightarrow \vec{x} = \vec{y}) \wedge F[\vec{y}/\vec{x}] \\ \rightarrow \\ \boldsymbol{\exists}\, \vec{y} : \Box(u = z \rightarrow \vec{x} = \vec{y}) \wedge G[\vec{y}/\vec{x}] \end{pmatrix} \end{pmatrix}$$

For arbitrary TLA* formulas F and G, a definition of $F \mathrel{-\!\!\triangleright} G$ is obtained from the definition of $\mathcal{C}(F) \mathrel{-\!\!\triangleright} \mathcal{C}(G)$ above through the equivalence

$$F \mathrel{-\!\!\triangleright} G \;\; \equiv \;\; (\mathcal{C}(F) \mathrel{-\!\!\triangleright} \mathcal{C}(G)) \wedge (F \rightarrow G)$$

Finally, we obtain a definition of $\overset{+}{\mathrel{-\!\!\triangleright}}$ from $\mathrel{-\!\!\triangleright}$ from theorem 5.6:

$$\begin{array}{rcl} \mathcal{C}(F) \overset{+}{\mathrel{-\!\!\triangleright}} \mathcal{C}(G) & \leftrightarrow & (\mathcal{C}(G) \mathrel{-\!\!\triangleright} \mathcal{C}(F)) \mathrel{-\!\!\triangleright} \mathcal{C}(G) \\ F \overset{+}{\mathrel{-\!\!\triangleright}} G & \leftrightarrow & (\mathcal{C}(F) \overset{+}{\mathrel{-\!\!\triangleright}} \mathcal{C}(G)) \wedge (F \rightarrow G) \end{array}$$

5.4 Application to CTL*

We demonstrate the flexibility of the abstract composition rule of section 5.2 by deriving a similar rule for assumption-commitment specifications in the branching-time temporal logic CTL* [38]. This application is somewhat more tentative than that of section 5.3, in part because of the expressiveness of CTL*, which allows many different styles of assumption-commitment specifications. We restrict attention to the fragment of CTL* where formulas are invariant under finite stuttering [61]—specifically, we do not allow the next-time operator.

Formulas of branching-time temporal logics are normally interpreted over infinite trees. They include state formulas, which are evaluated at a state in a tree, and path formulas, which are evaluated on a path in a tree. We write $M, s \models F$ if the state formula F is true at state s in tree M.

We extend the semantics of state formulas to finite trees. We say that M is a subtree of N, and write $M \sqsubseteq N$, if M has the same root as N and M's accessibility relation is included in N's. For a finite tree M and a state s, we write $M, s \models F$ if s is not a node of M or if there exists some infinite tree N such that $M \sqsubseteq N$ and $N, s \models F$.

When s is the root of M, we may simply say that F is true of M, and write $M \models F$. A specification is given by a state formula F; it describes the set of trees N such that $N \models F$. A state formula F is a safety property if $N \models F$ whenever $M \models F$ for all finite subtrees $M \sqsubseteq N$.

As in the linear-time case, we have the connectives $\mathcal{C}$, $\mathrel{-\!\!\triangleright}$, and $\overset{+}{\mathrel{-\!\!\triangleright}}$:

- If F is a state formula, then $\mathcal{C}(F)$ is a state formula, with $N, s \models \mathcal{C}(F)$ iff $M, s \models F$ holds for all finite subtrees $M \sqsubseteq N$.

- If F and G are state formulas, then $F \mathrel{-\!\!\triangleright} G$ is a state formula, where we define $N, s \models F \mathrel{-\!\!\triangleright} G$ iff for all (finite and infinite) subtrees M of N, if $M, s \models F$ then $M, s \models G$.

- If F and G are state formulas, then $F \overset{+}{\mathrel{-\!\!\triangleright}} G$ is a state formula, where we define $N, s \models F \overset{+}{\mathrel{-\!\!\triangleright}} G$ iff both:

  1. for all finite subtrees $M \sqsubseteq N$, if $T, s \models F$ holds for all $T \sqsubset M$ then $M, s \models G$, and
  2. $N, s \models F \mathrel{-\!\!\triangleright} G$.

Again, $\mathcal{C}(F)$ denotes the strongest safety property implied by F; and F is equivalent to $\mathcal{C}(F)$ iff F is a safety property. If N is a finite tree, then $N, s \models \mathcal{C}(F)$ iff $N, s \models F$. If F and G are safety properties then $F \mathrel{-\!\!\triangleright} G$ and $F \overset{+}{\mathrel{-\!\!\triangleright}} G$ are safety properties too. For all F and G, we have:

$$(F \mathrel{-\!\!\triangleright} G) \quad \rightarrow \quad (\mathcal{C}(F) \mathrel{-\!\!\triangleright} \mathcal{C}(G)) \wedge (F \rightarrow G) \tag{5.11}$$

$$(F \overset{+}{\mathrel{-\!\!\triangleright}} G) \quad \rightarrow \quad (\mathcal{C}(F) \overset{+}{\mathrel{-\!\!\triangleright}} \mathcal{C}(G)) \wedge (F \rightarrow G) \tag{5.12}$$

Note the differences with the corresponding definitions and results for TLA*. The differences arise because an infinite tree may have infinite proper subtrees, while the proper subsequences of an infinite sequence are all finite.

Finite trees yield another model of the abstract logic of section 5.2. Specifically, let Σ be the set of finite trees ordered by $\sqsubseteq$. Clearly, $\sqsubset$ is well-founded on Σ. To each state formula F corresponds the set $\mathcal{T}(F)$ of finite trees of which F is true; this is a downward-closed subset of Σ. For finite trees, we get the analogues of lemma 5.12.1 and 5.12.2. For safety properties, the analogue of 5.12.3 holds as well; this would not be true if we had allowed the next-time operator.

We represent the assumption-commitment specification with assumption A and commitment C by the formula $A \overset{+}{\mathrel{-\!\!\triangleright}} C$. We obtain the following theorem for specifications of the form $A \overset{+}{\mathrel{-\!\!\triangleright}} C$:

Theorem 5.15 *If the CTL* formulas A, C, A_i, C_i are safety properties and do not contain the next-time operator (for $i \in I$), then the following formula is valid:*

$$\begin{array}{l} (A \wedge \bigwedge_{i \in I} C_i \mathrel{-\!\!\triangleright} \bigwedge_{i \in I} A_i) \wedge (A \overset{+}{\mathrel{-\!\!\triangleright}} (\bigwedge_{i \in I} C_i \mathrel{-\!\!\triangleright} C)) \\ \quad \mathrel{-\!\!\triangleright} \\ (\bigwedge_{i \in I} (A_i \overset{+}{\mathrel{-\!\!\triangleright}} C_i) \mathrel{-\!\!\triangleright} (A \overset{+}{\mathrel{-\!\!\triangleright}} C)) \end{array}$$

The proof uses exactly the same reasoning as the corresponding proof for TLA*. Going beyond safety properties, we can obtain a CTL* analogue for Theorem 5.14; the proof is basically the same as that of Theorem 5.14, and relies on the validity of $F \rightarrow \mathcal{C}(F)$, the implication-like properties of $\rightarrowtriangle$, and (5.11) and (5.12).

In branching-time temporal logics, the composition of modules does not in general implement the conjunction of the specifications of the modules. The application of our theorems as composition rules will therefore require additional arguments. This complication is not unique to our work; several authors [50, 41] have advocated restricting commitments to the fragment $\forall$CTL* in order to ensure that specifications are preserved by composition.

Josko [50, 51] has suggested representing an assumption-commitment specification as a pair (A, C) where A is a linear-time formula and C is a branching-time formula; Vardi [102] has studied the complexity of model-checking for specifications of this form. With Josko's definitions, a tree M satisfies a specification (A, C) with assumption A and commitment C iff $M' \models C$ where M' is the subtree of M that consists of those paths that satisfy A. Instead of Josko's (A, C), we can write $(\forall A) \overset{+}{\rightarrowtriangle} C$, which is similar but logically stronger. The similarity between (A, C) and $(\forall A) \overset{+}{\rightarrowtriangle} C$ is even closer under the additional hypotheses of Josko's rules for dealing with mutual dependencies [51].

5.5 Related work

We have studied specifications in a general logical framework. We then inferred concrete proof rules for composing specifications from general logical facts. We believe that this approach explains some of the principles that underly the rules and helps in comparing rules.

Our abstract logic borrows from that of Abadi and Plotkin [8]. In particular, we take the idea of using intuitionistic reasoning for assumption-commitment specifications. However, for the sake of simplicity and generality, we do not adopt some non-standard constructs of that logic (for example, "constrains at most").

Cau, Collette, and Xu have given another interesting, unifying perspective on rules for composition [105, 27]. Their work treats abstract rules semantically; concurrent processes, with either shared variables or message passing, are then embedded in a common semantic structure based on labelled sequences. This structure could provide a model for our logic (much like the model of section 5.3). In comparison, our abstract treatment of composition is primarily syntactic. We resort to semantic reasoning about computations only in applications to particular formalisms. This approach enables us to consider non-linear well-founded structures, for example in the context of branching-time logics.

Both of the applications described in detail in this paper are for temporal logics. However, our approach is not intrinsically limited to temporal logics: we have also used it on specifications of stream-processing functions [22, 24, 95]. An assumption-commitment specification for a stream-processing function gives a property of the result of the function under assumptions about the inputs of the functions. Inductive reasoning arises when the function is defined as a fixpoint. We can represent that reasoning in our abstract logic, and thus prove variants of the proof rules of Stølen et al. [95].

The first compositional proof rules for languages based on synchronous message passing and shared variables were introduced by Misra and Chandy [78], Jones [47], and for temporal logic by Barringer et al [16]. The interest in compositional reasoning culminated in the 1980s and early 1990s when the body of published research on compositionality became overwhelming. An excellent survey of the state of the art can be found in the proceedings of a symposium devoted to the topic [34]. The excitement was perhaps mitigated later by the observation that examples of assumption-commitment specifications of reusable components are hard to find in practical applications, much less proofs about such components [64]. Shankar [90] has advocated another approach to compositional reasoning where systems are decomposed as needed during verification. Following Shankar, no general assumption-commitment specification of a component need be written in advance; this may make compositional verification more practical, although it does not provide specifications of reusable library components.

Chapter 6

Abstraction techniques

6.1 Overview

Abstraction techniques allow to relate a "concrete" system Φ and an "abstract" system $\overline{\Phi}$ such that properties $\overline{P}$ established for $\overline{\Phi}$ imply corresponding properties P for Φ. Usually, Φ is a large or infinite-state system, whereas the state space of $\overline{\Phi}$ should be small enough for automatic verification techniques such as model checking to be feasible. The correspondence between Φ and $\overline{\Phi}$ should be established by "local" reasoning about states and state transitions; it can be supported by interactive theorem provers or by abstract interpretation techniques. In this sense, abstraction serves as an interface between automatic and interactive verification techniques. It is important to note that the abstract system $\overline{\Phi}$ will be determined by both Φ and the property P of interest; in particular, verifying different properties of the same system usually requires using different abstractions.

In this chapter we outline a theory of abstraction for TLA* specifications that offers a viable alternative to the traditional, deductive approach to system verification. The main promise of abstraction-based verification is that much of the tedium that is associated with deductive verification can be left to the model checker. One should therefore expect the most substantial gains when abstraction techniques are used to verify liveness properties. Unfortunately, most naïve approaches to abstraction are not well suited to support reasoning about liveness properties. This is because the identification of concrete states that is desirable to reduce the size of the abstract system is likely to introduce cycles in the abstract state space that interfere with establishing liveness properties. Consequently, most of the literature on abstraction concentrates on safety properties, for which false cycles are less problematic. The main challenge is then to automate the abstraction process as far as possible, and impressive progress in this area has been reported, for example, in the SAL [19] and Bandera [30] projects.

In contrast, our main focus, after introducing a basic theory of homomorphic abstraction in a logical setting, will be on techniques that allow to strengthen abstractions such as to break false cycles. We will identify two main techniques: first, we will show how arguments based on well-founded orderings that are routinely used in deductive verification can be represented in the context of homomorphic abstractions. Second, we describe a technique that, paradoxically, weakens the fairness conditions given at the concrete level before computing the abstraction, in order to obtain stronger fairness conditions at the abstract level.

We will in the following, mainly for simplicity of presentation, ignore the extra complications due to invariance under stuttering, and work in a "raw" version of TLA* where $\Box A$ is a pre-formula, for any pre-formula A. Consequently, the semantics of quantification in "raw" TLA* is based on the relation $=_x$ instead of $\approx_x$. In the following chapter 7 we will study an alternative presentation of abstractions in terms of so-called predicate diagrams where stuttering invariance will again be taken into account.

6.2 Homomorphic abstraction

The specifications Φ and $\overline{\Phi}$ are both represented as (raw) TLA* formulas. The state components are represented by (tuples of) flexible variables, say x for the concrete level and $\overline{x}$ for the abstract level. Without loss of generality, we assume the tuples x and $\overline{x}$ to be disjoint. The abstraction function (or tuple of functions) h that maps x into $\overline{x}$ is represented by a term with free variables among x. The basic principle of proofs by abstraction is reflected in the proof rule[1]

$$\frac{\begin{array}{c}\Phi \wedge \Box A \wedge \Box(\overline{x} = h) \rightarrow \overline{\Phi} \\ \overline{\Phi} \rightarrow \overline{P} \\ \overline{P} \wedge \Box A \wedge \Box(\overline{x} = h) \rightarrow P\end{array}}{\Phi \wedge \Box A \rightarrow P} \quad (\overline{x} \notin \mathit{FFV}(\Phi, A, P)) \tag{6.1}$$

The second hypothesis asserts that the abstract system $\overline{\Phi}$ satisfies the abstract property $\overline{P}$; when $\overline{\Phi}$ describes a finite-state system of reasonably small size, this can be established using model checking. On the other hand, the first and third hypotheses of rule (6.1) relate the specifications and properties at the abstract and concrete levels. The linking invariant A is usually built from invariants and actions implied by the next-state relation of the system specification Φ. Formally, it can be any formula or pre-formula at the concrete level, that is, without free occurrences of $\overline{x}$. It is typically needed only for the proof of the third premise, which requires a property at the concrete level to be inferred from a property at the abstract level.

[1] By $\overline{x} = h$ we denote the componentwise equality of the tuples $\overline{x}$ and h.

Proposition 6.1 *Rule (6.1) is sound.*

Proof. The premises of (6.1) immediately establish

$$\Phi \wedge \Box A \wedge \Box(\overline{x} = h) \rightarrow P$$

From the assumption that the variables $\overline{x}$ do not occur free in Φ, A, and P and standard quantifier rules we obtain

$$\Phi \wedge \Box A \wedge (\exists \overline{x} : \Box(\overline{x} = h)) \rightarrow P$$

Now the conclusion follows from the tautology $\Box(h = h)$, which implies the validity of the conjunct $\exists \overline{x} : \Box(\overline{x} = h)$. Q.E.D.

Taken at face value, rule (6.1) may not seem very promising: even if its second premise can be shown algorithmically, it is not clear why the two remaining premises should be easier to establish than the conclusion itself. Instead of leaving these premises to standard deductive verification, we exploit the intuition that the abstract specification is a "homomorphic image" of the concrete. In our logic-based setting this means that the formulas at the two levels have similar shape. We show that $\overline{\Phi}$ and $\overline{P}$ can be constructed systematically from abstractions of their elementary constituents (i.e., state and transition formulas).

Definition 6.2 *Let h be a tuple of state functions and A be an action.*

1. *Given an action N, actions N^- and N^+ are called* weakening *and* strengthening abstractions *of N with respect to h and A if the following transition formulas are valid:*

$$A \wedge \overline{x} = h \wedge \overline{x}' = h' \rightarrow (N \rightarrow N^-)$$
$$A \wedge \overline{x} = h \wedge \overline{x}' = h' \rightarrow (N^+ \rightarrow N)$$

2. *Given a (pre-)formula F, (pre-)formulas F^- and F^+ are called* weakening *and* strengthening abstractions *of F with respect to h and A if the following (pre-)formulas are valid:*

$$\Box A \wedge \Box(\overline{x} = h) \rightarrow (F \rightarrow F^-)$$
$$\Box A \wedge \Box(\overline{x} = h) \rightarrow (F^+ \rightarrow F)$$

Remark. Note that definition 6.2 implies that there are complete lattices of weakenings and strengthenings of any given formula. Indeed, while **true** and **false** are, respectively, the weakest weakening and strongest strengthening of any formula, the strongest weakening N^-_{max} and weakest strengthening N^+_{min} of an action formula N with respect to h and A can be defined as

$$\begin{array}{lcl} N^-_{max} & \equiv & \exists x, x' : A \wedge \overline{x} = h \wedge \overline{x}' = h' \wedge N \\ N^+_{min} & \equiv & \forall x, x' : A \wedge \overline{x} = h \wedge \overline{x}' = h' \rightarrow N \end{array}$$

The corresponding definitions for a temporal formula F are analogous, using quantification over flexible variables. Some authors [53, 28] take similar formulas as the *definitions* of abstraction, implying that weakenings and strengthenings of a given formula are functionally determined. However, the quantifiers will have to be eliminated before model checking can be applied, and the definitions above allow more flexibility in choosing abstractions appropriate to the problem at hand. In fact, our notion of a weakening abstraction subsumes both the notions of "abstraction" and "approximation" from [28].

Definition 6.2 requires the *validity* of certain formulas in order to show that a formula is indeed an abstraction. Consequently, any free rigid variables that occur in the formulas at the concrete or abstract level are implicitly universally quantified. The definitions of temporal weakening and strengthening abstractions have been chosen in view of the first and third premises of rule (6.1), which can now be rephrased as

$$\frac{\Phi^- \rightarrow P^+}{\Phi \wedge \Box A \rightarrow P} \tag{6.2}$$

for any weakening Φ^- of Φ and strengthening P^+ of P for given h and A.

For practical applications, it is important to be able to reduce the proof obligation for temporal weakenings and strengthenings to those at the action level. The following proposition shows that abstractions of complex formulas can be obtained from those of their constituents.

Proposition 6.3 *Assume given weakenings F^-, G^- and strengthenings F^+, G^+ of transition or temporal (pre-)formulas F and G. Abstractions of complex formulas can be computed according to table 6.1.*

In the cases of $\Box[F]_v$ and $\Diamond\langle F\rangle_v$ we write $\overline{v}$ to denote the flexible variables that correspond to v at the abstract level. Note that for these cases our rules may in general yield raw formulas; if we were working in "pure" TLA* we would have to apply further weakening or strengthening to obtain well-formed formulas. The correctness of the transformations follows easily from the definitions of weakenings and strengthenings and has been verified using the interactive theorem prover Isabelle [83].

formula	weakening	strengthening
$\neg F$	$\neg F^+$	$\neg F^-$
$F \to G$	$F^+ \to G^-$	$F^- \to G^+$
$F \wedge G$	$F^- \wedge G^-$	$F^+ \wedge G^+$
$F \vee G$	$F^- \vee G^-$	$F^+ \vee G^+$
$\circ F$	$\circ F^-$	$\circ F^+$
$\Box F$	$\Box F^-$	$\Box F^+$
$\Diamond F$	$\Diamond F^-$	$\Diamond F^+$
$\Box[F]_v$	$\Box[F^-]_{\overline{v}}$	$\Box(F^+ \vee (v' = v)^+)$
$\Diamond\langle F\rangle_v$	$\Diamond(F^- \wedge (v' \neq v)^-)$	$\Diamond\langle F^+\rangle_{\overline{v}}$
$\forall y.F$	$\forall y.F^-$	$\forall y.F^+$
$\exists y.F$	$\exists y.F^-$	$\exists y.F^+$

Table 6.1: Structural abstraction rules

Formulas that involve quantification over flexible variables require more careful analysis. We first consider the enabledness predicate of actions, which is required to compute abstractions of fairness conditions.

Lemma 6.4 *Let N and A be action formulas and h be a state function.*

1. *If N^- is a weakening abstraction of N w.r.t. A and h and*

$$A \wedge \text{ENABLED } N \to \text{ENABLED } (A \wedge N \wedge \overline{x}' = h')$$

then ENABLED N^- *is a weakening of* ENABLED N *w.r.t. A and h.*

2. *If N^+ is a strengthening abstraction of N w.r.t. A and h and*

$$A \wedge \overline{x} = h \wedge \text{ENABLED } N^+ \to \text{ENABLED } (A \wedge \overline{x}' = h' \wedge N^+)$$

then ENABLED N^+ *is a strengthening of* ENABLED N *w.r.t. A and h.*

Proof. If N^- is a weakening of N, we have

$$A \wedge \overline{x} = h \wedge \overline{x}' = h' \wedge N \to N^-$$

Existential quantification on both sides of the implication yields

$$\text{ENABLED } (A \wedge \overline{x} = h \wedge \overline{x}' = h' \wedge N) \to \text{ENABLED } N^-$$

This formula may be equivalently transformed by observing that the quantified variables do not occur in the conjunct $\overline{x} = h$, and that the variables x' do not occur in N^- to the formula

$$\overline{x} = h \wedge \text{ENABLED } (A \wedge \overline{x}' = h' \wedge N) \rightarrow \text{ENABLED } N^-$$

Using the assumption of the lemma we obtain

$$A \wedge \overline{x} = h \wedge \text{ENABLED } N \rightarrow \text{ENABLED } N^-$$

which shows that ENABLED N^- is indeed a weakening of ENABLED N.

The proof of assertion (2) is similar. Q.E.D.

When computing abstractions of formulas with temporal quantification over flexible variables, the underlying abstraction of the formula in the scope of the quantifier will be computed w.r.t. a different abstraction function and linking invariant than that of the quantified formula because the state space contains an extra variable. In the following lemma we explicitly note the variables that may occur free in the different formulas.

Lemma 6.5 *Let A, B be (pre-)formulas such that y does not occur free in A, let g, h be state functions such that y does not occur free in g, and let F be any (pre-)formula.*

1. *If F^- is a weakening abstraction of F w.r.t. $A \wedge B$ and (g, h) and*

$$\Box A \wedge F \rightarrow \boldsymbol{\exists}\, y : \Box B \wedge F$$

then $\boldsymbol{\exists}\, \overline{y} : F^-$ is a weakening of $\boldsymbol{\exists}\, y : F$ w.r.t. A and g.

2. *If F^+ is a strengthening abstraction of F w.r.t. $A \wedge B$ and (g, h) and*

$$\Box A \wedge \Box(\overline{x} = g) \wedge F^+ \rightarrow \boldsymbol{\exists}\, y : \Box B \wedge \Box(\overline{y} = h)$$

then $\boldsymbol{\exists}\, \overline{y} : F^+$ is a strengthening of $\boldsymbol{\exists}\, y : F$ w.r.t. A and g.

Proof.

1. Since F^- is a weakening of F, we have

$$\Box(A \wedge B) \wedge \Box(\overline{x} = g \wedge \overline{y} = h) \wedge F \rightarrow F^-$$

Existential quantification on both sides of the implication and simplification according to the assumptions on the variable occurrences yields

$$\Box A \wedge \Box(\overline{x} = g) \wedge (\boldsymbol{\exists}\, y, \overline{y} : \Box B \wedge \Box(\overline{y} = h) \wedge F) \rightarrow \boldsymbol{\exists}\, \overline{y} : F^-$$

On the other hand, we obtain

$$\Box A \wedge (\exists\, y : F) \rightarrow \exists\, y, \overline{y} : \Box B \wedge \Box(\overline{y} = h) \wedge F$$

from the assumption and the fact that y does not occur free in A and $\overline{y}$ does not occur free in B or F. Together, we conclude

$$\Box A \wedge \Box(\overline{x} = g) \wedge (\exists\, y : F) \rightarrow \exists\, \overline{y} : F^-$$

which proves that $\exists\, \overline{y} : F^-$ is a weakening of $\exists\, y : F$.

2. Since F^+ is a strengthening of F, we have

$$\Box(A \wedge B) \wedge \Box(\overline{x} = g \wedge \overline{y} = h) \wedge F^+ \rightarrow F$$

Quantification on both sides of the implication and simplification yields

$$\Box A \wedge \Box(\overline{x} = g) \wedge (\exists\, y, \overline{y} : \Box B \wedge \Box(\overline{y} = h) \wedge F^+) \rightarrow \exists\, y : F$$

On the other hand, we obtain

$$\Box A \wedge \Box(\overline{x} = g) \wedge (\exists\, \overline{y} : F^+) \rightarrow \exists\, y, \overline{y} : \Box B \wedge \Box(\overline{y} = h) \wedge F^+$$

from the assumption, quantifier rules, and the assumptions about the occurrences of y and $\overline{y}$. Together, we conclude

$$\Box A \wedge \Box(\overline{x} = g) \wedge (\exists\, \overline{y} : F^+) \rightarrow \exists\, y : F$$

which proves that $\exists\, \overline{y} : F^+$ is a strengthening of $\exists\, y : F$. Q.E.D.

Rule (6.2) and the preceding results, which imply that abstractions of formulas can be computed from abstractions of their elementary constituents, suggest the following methodology for verification by abstraction:

- Establish abstractions of the individual system actions and the predicates that occur in the formula to be verified. In general, this amounts to verifiying the proof obligations of definition 6.2 for nontemporal formulas (state predicates and actions), and can be supported by standard interactive theorem provers.

- Construct abstractions Φ^- and P^+ of the system specification Φ and the property P according to the structure of the formulas.

- Establish the validity of $\Phi^- \rightarrow P^+$ by algorithmic verification techniques.

We will now illustrate this approach at the hand of an example and then consider techniques that overcome common problems when verifying liveness properties.

6.3 Example: Bakery Algorithm

Lamport's "Bakery" algorithm [60] is a well-known protocol that solves the mutual-exclusion problem. Figure 6.1 contains representations of a version of the Bakery algorithm for two processes based on atomic actions, both in a pseudo-code version and as a TLA specification (in interleaving style).[2] We wish to establish the mutual-exclusion property as well as the liveness of the algorithm, that is, we want to prove

$$Bakery \rightarrow \Box\neg(pc_1 = \text{"l3"} \wedge pc_2 = \text{"m3"}) \tag{6.3}$$
$$Bakery \rightarrow (pc_1 = \text{"l2"} \leadsto pc_1 = \text{"l3"}) \tag{6.4}$$

Observe that the two-process version that we are considering here is already an infinite-state system because the values of the "ticket" variables t_1 and t_2 can grow beyond any bound in a run where each of the processes executes its Req action while the other process is in its critical section. It can therefore not be verified using standard model checking techniques. A natural abstraction function is suggested by inspecting the predicates that appear in the guards and in the formulas: whereas the program counters are left unabstracted, we record for each t_i whether it is zero or not, and we remember whether $t_1 < t_2$ holds. Formally, the abstract state space is given by the flexible variables $\overline{pc}_1$, $\overline{pc}_2$, $\overline{t}_1$, $\overline{t}_2$, and $\overline{less}$. The variables at the abstract and the concrete levels are related by the abstraction function h expressed by the state predicate

$$\begin{array}{rcl} hom & \equiv & \wedge\ \overline{pc}_1 = pc_1 \wedge \overline{pc}_2 = pc_2 \wedge \overline{t}_1 = \mathrm{sg}(t_1) \wedge \overline{t}_2 = \mathrm{sg}(t_2) \\ & & \wedge\ \overline{less} = \textbf{if } t_1 \leq t_2 \textbf{ then } 1 \textbf{ else } 2 \end{array}$$

where sg denotes the sign function. As the linking invariant we choose the state predicate

$$A \equiv t_1 \in Nat \wedge t_2 \in Nat$$

We note that the invariant $\Box(t_1 \in Nat \wedge t_2 \in Nat)$ is easily established for system $Bakery$, for example by static analysis of the specification text.

Lemma 6.6 *The specification $\overline{Bakery}$ shown in figure 6.2 is a weakening abstraction of $Bakery$ w.r.t. h and A.*

[2] The original, non-atomic version of the Bakery algorithm could be verified along the same lines. We have chosen the atomic version because it generates fewer proof obligations.

integer $t_1, t_2 = 0$;
cobegin

loop		**loop**
l_0 : (* noncritical *)		m_0 : (* noncritical *)
$l_1 : t_1 := t_2 + 1$;		$m_1 : t_2 := t_1 + 1$;
l_2 : **await** $t_2 = 0 \lor t_1 \leq t_2$;	$\|$	m_2 : **await** $t_1 = 0 \lor \neg(t_1 \leq t_2)$;
l_3 : (* critical *)		m_3 : (* critical *)
$l_4 : t_1 := 0$		$m_4 : t_2 := 0$
endloop		**endloop**

coend

(a) Pseudocode representation

$$
\begin{array}{rcl}
Init & \equiv & pc_1 = \text{“l0”} \land pc_2 = \text{“m0”} \land t_1 = 0 \land t_2 = 0 \\
NCrit_1 & \equiv & pc_1 = \text{“l0”} \land pc_1' = \text{“l1”} \land \text{UNCHANGED } \langle pc_2, t_1, t_2 \rangle \\
Req_1 & \equiv & pc_1 = \text{“l1”} \land pc_1' = \text{“l2”} \land t_1' = t_2 + 1 \land \text{UNCHANGED } \langle pc_2, t_2 \rangle \\
Try_1 & \equiv & \land\ pc_1 = \text{“l2”} \land (t_2 = 0 \lor t_1 \leq t_2) \\
 & & \land\ pc_1' = \text{“l3”} \land \text{UNCHANGED } \langle pc_2, t_1, t_2 \rangle \\
Crit_1 & \equiv & pc_1 = \text{“l3”} \land pc_1' = \text{“l4”} \land \text{UNCHANGED } \langle pc_2, t_1, t_2 \rangle \\
Exit_1 & \equiv & pc_1 = \text{“l4”} \land pc_1' = \text{“l0”} \land t_1' = 0 \land \text{UNCHANGED } \langle pc_2, t_2 \rangle \\
NCrit_2 & \equiv & pc_2 = \text{“m0”} \land pc_2' = \text{“m1”} \land \text{UNCHANGED } \langle pc_1, t_1, t_2 \rangle \\
Req_2 & \equiv & pc_2 = \text{“m1”} \land pc_2' = \text{“m2”} \land t_2' = t_1 + 1 \land \text{UNCHANGED } \langle pc_1, t_1 \rangle \\
Try_2 & \equiv & \land\ pc_2 = \text{“m2”} \land (t_1 = 0 \lor \neg(t_1 \leq t_2)) \\
 & & \land\ pc_2' = \text{“m3”} \land \text{UNCHANGED } \langle pc_1, t_1, t_2 \rangle \\
Crit_2 & \equiv & pc_2 = \text{“m3”} \land pc_2' = \text{“m4”} \land \text{UNCHANGED } \langle pc_1, t_1, t_2 \rangle \\
Exit_2 & \equiv & pc_2 = \text{“m4”} \land pc_2' = \text{“m0”} \land t_2' = 0 \land \text{UNCHANGED } \langle pc_1, t_1 \rangle \\
Next & \equiv & \lor\ NCrit_1 \lor Req_1 \lor Try_1 \lor Crit_1 \lor Exit_1 \\
 & & \lor\ NCrit_2 \lor Req_2 \lor Try_2 \lor Crit_2 \lor Exit_2 \\
vars & \equiv & \langle pc_1, pc_2, t_1, t_2 \rangle \\
Bakery & \equiv & Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(Next)
\end{array}
$$

(b) TLA specification

Figure 6.1: Bakery algorithm for two processes (interleaving representation)

$$
\begin{array}{rcl}
\overline{Init} & \equiv & \overline{pc}_1 = \text{“l0”} \wedge \overline{pc}_2 = \text{“m0”} \wedge \overline{t}_1 = 0 \wedge \overline{t}_2 = 0 \wedge \overline{less} = 2 \\
\overline{NCrit}_1 & \equiv & \overline{pc}_1 = \text{“l0”} \wedge \overline{pc}'_1 = \text{“l1”} \wedge \text{UNCHANGED } \langle \overline{pc}_2, \overline{t}_1, \overline{t}_2, \overline{less} \rangle \\
\overline{Req}_1 & \equiv & \wedge\, \overline{pc}_1 = \text{“l1”} \wedge \overline{pc}'_1 = \text{“l2”} \\
 & & \wedge\, \overline{t}'_1 = 1 \wedge \overline{less}' = 2 \wedge \text{UNCHANGED } \langle \overline{pc}_2, \overline{t}_2 \rangle \\
\overline{Try}_1 & \equiv & \wedge\, \overline{pc}_1 = \text{“l2”} \wedge (\overline{t}_2 = 0 \vee \overline{less} = 1) \\
 & & \wedge\, \overline{pc}'_1 = \text{“l3”} \wedge \text{UNCHANGED } \langle \overline{pc}_2, \overline{t}_1, \overline{t}_2, \overline{less} \rangle \\
\overline{Crit}_1 & \equiv & \overline{pc}_1 = \text{“l3”} \wedge \overline{pc}'_1 = \text{“l4”} \wedge \text{UNCHANGED } \langle \overline{pc}_2, \overline{t}_1, \overline{t}_2, \overline{less} \rangle \\
\overline{Exit}_1 & \equiv & \wedge\, \overline{pc}_1 = \text{“l4”} \wedge \overline{pc}'_1 = \text{“l0”} \\
 & & \wedge\, \overline{t}'_1 = 0 \wedge \overline{less}' = \textbf{if } \overline{t}_2 = 0 \textbf{ then } 2 \textbf{ else } 1 \\
 & & \wedge\, \text{UNCHANGED } \langle \overline{pc}_2, \overline{t}_2 \rangle \\
\overline{NCrit}_2 & \equiv & \overline{pc}_2 = \text{“m0”} \wedge \overline{pc}'_2 = \text{“m1”} \wedge \text{UNCHANGED } \langle \overline{pc}_1, \overline{t}_1, \overline{t}_2, \overline{less} \rangle \\
\overline{Req}_2 & \equiv & \wedge\, \overline{pc}_2 = \text{“m1”} \wedge \overline{pc}'_2 = \text{“m2”} \\
 & & \wedge\, \overline{t}'_2 = 1 \wedge \overline{less}' = 1 \wedge \text{UNCHANGED } \langle \overline{pc}_1, \overline{t}_1 \rangle \\
\overline{Try}_2 & \equiv & \wedge\, \overline{pc}_2 = \text{“m2”} \wedge (\overline{t}_1 = 0 \vee \overline{less} = 2) \\
 & & \wedge\, \overline{pc}'_2 = \text{“m3”} \wedge \text{UNCHANGED } \langle \overline{pc}_1, \overline{t}_1, \overline{t}_2, \overline{less} \rangle \\
\overline{Crit}_2 & \equiv & \overline{pc}_2 = \text{“m3”} \wedge \overline{pc}'_2 = \text{“m4”} \wedge \text{UNCHANGED } \langle \overline{pc}_1, \overline{t}_1, \overline{t}_2, \overline{less} \rangle \\
\overline{Exit}_2 & \equiv & \wedge\, \overline{pc}_2 = \text{“m4”} \wedge \overline{pc}'_2 = \text{“m0”} \\
 & & \wedge\, \overline{t}'_2 = 0 \wedge \overline{less}' = 2 \wedge \text{UNCHANGED } \langle \overline{pc}_1, \overline{t}_1 \rangle \\
\overline{Next} & \equiv & \vee\, \overline{NCrit}_1 \vee \overline{Req}_1 \vee \overline{Try}_1 \vee \overline{Crit}_1 \vee \overline{Exit}_1 \\
 & & \vee\, \overline{NCrit}_2 \vee \overline{Req}_2 \vee \overline{Try}_2 \vee \overline{Crit}_2 \vee \overline{Exit}_2 \\
\overline{vars} & \equiv & \langle \overline{pc}_1, \overline{pc}_2, \overline{t}_1, \overline{t}_2, \overline{less} \rangle \\
\overline{Bakery} & \equiv & \overline{Init} \wedge \Box[\overline{Next}]_{\overline{vars}} \wedge \mathrm{WF}_{\overline{vars}}(\overline{Next})
\end{array}
$$

Figure 6.2: Abstraction of specification $Bakery$

Proof (sketch). It is easy to see that the initial predicate and each of the actions of $\overline{Bakery}$ are weakenings of the initial predicate and the corresponding actions of $Bakery$. By proposition 6.3 it follows that $\overline{Init} \wedge \Box[\overline{Next}]_{\overline{vars}}$ is a weakening of $Init \wedge \Box[Next]_{vars}$.

For the fairness condition, it suffices to prove that $\langle \overline{Next} \rangle_{\overline{vars}}$ is a weakening of $\langle Next \rangle_{vars}$ and that ENABLED $\langle \overline{Next} \rangle_{\overline{vars}}$ is a strengthening of ENABLED $\langle Next \rangle_{vars}$ w.r.t. A and h. (The enabledness predicate occurs negatively in the definition of the weak fairness condition.) For the first assertion we argue as follows:

$$
\begin{array}{ll}
& A \wedge \langle Next \rangle_{vars} \wedge hom \wedge hom' \\
\Rightarrow & \quad [\text{by definition of } Next] \\
& Next \wedge (pc_1' \neq pc_1 \vee pc_2' \neq pc_2) \wedge hom \wedge hom' \\
\Rightarrow & \quad [\overline{Next} \text{ is a weakening of } Next\text{, definition of } hom] \\
& \overline{Next} \wedge (\overline{pc}_1' \neq \overline{pc}_1 \vee \overline{pc}_2' \neq \overline{pc}_2) \\
\Rightarrow & \langle \overline{Next} \rangle_{\overline{vars}}
\end{array}
$$

On the other hand, we note that ENABLED $\langle Next \rangle_{vars}$ equivales **true**, because at least one process can make progress in any state: If both processes were blocked, they would both have to be at their **await** statements with both guards false, which is clearly impossible. Therefore ENABLED $\langle \overline{Next} \rangle_{\overline{vars}}$ is trivially a strengthening of ENABLED $\langle Next \rangle_{vars}$ w.r.t. A and h.

The lemma has been proven formally using Isabelle. Q.E.D.

On the other hand, the properties (6.3) and (6.4) that we wish to verify of specification $Bakery$ are expressed in terms of pc_1 and pc_2, which are left unabstracted in $\overline{Bakery}$. Obviously, the analogous formulas

$$
\begin{array}{l}
\Box\neg(\overline{pc}_1 = \text{“l3”} \wedge \overline{pc}_2 = \text{“m3”}) \\
\quad \overline{pc}_1 = \text{“l2”} \leadsto \overline{pc}_1 = \text{“l3”}
\end{array}
$$

at the abstract level are strengthenings of the properties of interest, and by rule (6.2) it suffices to show that they are implied by $\overline{Bakery}$. Now, specification $\overline{Bakery}$ describes a finite-state system and is therefore amenable to model checking. We have encoded specification $\overline{Bakery}$ as a fair transition system in STeP [69] and have used STeP's model checking component to automatically verify both properties.

As a somewhat curious aside, observe that the abstraction $\overline{Bakery}$ can be seen as a slight variation of Peterson's mutual-exclusion algorithm; the successful run of the model checker shows that it is a correct solution to the mutual-exclusion problem. Of course, the original Bakery algorithm ensures that each variable is written by only one process. This attractive feature is lost in $\overline{Bakery}$, since the variable $\overline{less}$ is updated by both processes.

Comparing the effort involved in our abstraction-based proof and in a traditional deductive verification attempt we observe the following:

- Finding the abstraction function was trivial for this example, because the values and predicates of interest were indicated by the state predicates that appear in the specification and the formulas. In contrast, standard deductive verification would have required to invent the auxiliary invariant

$$
\begin{array}{l}
\wedge \vee pc_1 \in \{\text{“l0”}, \text{“l1”}\} \wedge t_1 = 0 \\
\quad \vee pc_1 \in \{\text{“l2”}, \text{“l3”}, \text{“l4”}\} \wedge t_1 \in Nat \setminus \{0\} \\
\wedge \vee pc_2 \in \{\text{“m0”}, \text{“m1”}\} \wedge t_2 = 0 \\
\quad \vee pc_2 \in \{\text{“m2”}, \text{“m3”}, \text{“m4”}\} \wedge t_2 \in Nat \setminus \{0\} \\
\wedge pc_1 \in \{\text{“l3”}, \text{“l4”}\} \rightarrow t_2 = 0 \vee t_1 \leq t_2 \\
\wedge pc_2 \in \{\text{“m3”}, \text{“m4”}\} \rightarrow t_1 = 0 \vee \neg(t_1 \leq t_2)
\end{array}
$$

- Verifying the correctness of the abstraction could be reduced to nontemporal reasoning about single system actions. This effort is comparable to verifying the inductive invariant given above.

- No extra effort was needed for the verification of the liveness property. In contrast, deductive verification would have required to consider the different possible locations of the two processes and to show that some progress towards the critical section was ensured in every case.

We observe that most of the benefit gained from using abstraction was in the verification of the liveness property, somewhat in contrast to conventional wisdom that holds abstraction to be most adequate for the verification of safety properties. On the other hand, not all specifications lend themselves to abstraction as easily as our example. It therefore appears promising to study the limitations of abstraction techniques for the verification of liveness properties, and how these can be overcome.

6.4 Augmented homomorphic abstraction

An abstraction function is useful only if the abstract state space is significantly smaller than that of the original system. However, there is a price to pay for this desired reduction: if two states s and t that are related by an action of the concrete system are mapped to the same state $h(s)$, the abstract model contains a loop where the concrete model does not. More generally, the abstract system is likely to contain many more cycles than the concrete one. Whereas the presence of additional cycles does not hinder the verification of safety properties, it will interfere with the verification of many liveness properties. In concrete applications this problem manifests itself in two ways:

- First, the fairness assumptions associated with the abstract specification may be insufficient to rule out all undesired cycling behaviors. In that case,

$\Phi^- \to P^+$ cannot be proven, and the model checker produces a counterexample over the abstract system that has no counterpart over the concrete system. In this case, we need to *strengthen the abstract specification* Φ^-, either by using a finer abstraction function or by adding additional liveness hypotheses that must be justified from the concrete specification Φ.

- Second, the fairness assumptions associated with the concrete specification may be too strong to obtain useful weakening abstractions (i.e., different from **true**). In this case, we have to *weaken the concrete specification* Φ by replacing its liveness assumption with one that produces a better weakening abstraction. This technique can be explained by a form of non-monotonicity in computing abstractions, which is formally due to the fact that the enabledness predicate occurs negatively in the definition of a fairness condition, whereas the action itself occurs positively.

Because we do not formally distinguish between specifications and properties, both principles can be abstractly represented by the proof rule

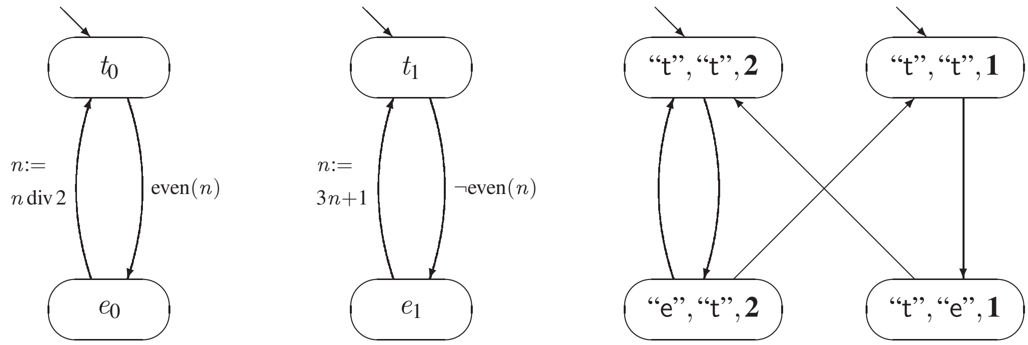
$$\frac{\Phi \to H \qquad \Phi^- \wedge H^- \to P^+}{\Phi \to P} \tag{6.5}$$

that can be used in place of rule (6.2). In this rule, H can either represent an auxiliary property used to strengthen the abstract liveness assumption or a weaker specification, and H^- denotes a weakening abstraction of H. The soundness of rule (6.5) is obvious by simple propositional reasoning and proposition 6.3.

According to the basic principle of reducing temporal proof obligations to nontemporal reasoning, the premise $\Phi \to H$ of rule (6.5) should be reduced to transition-level proof obligations by general principles. The following sections will present two instances that we have found useful: one technique makes use of well-founded orderings in the concrete state space to strengthen the liveness assumption of the abstract specification, whereas the other represents a systematic way to weaken fairness assumptions that is particularly useful for the verification of parameterized systems.

6.5 Abstractions and well-founded orderings

Consider the (somewhat artificial) "dining mathematicians" mutual-exclusion protocol shown in figure 6.3 that has been introduced by Dams et al [32]. The system consists of two processes P_0 and P_1 that communicate via the shared variable n, initialized to some positive natural number. The processes can either be in a "thinking" or "eating" state, initially they are "thinking". Process P_0 may eat



(a) State-transition diagrams

$$
\begin{aligned}
\mathit{Init} &\equiv n \in \mathit{Nat} \setminus \{0\} \land c_0 = \text{“t”} \land c_1 = \text{“t”} \\
\mathit{Eat}_0 &\equiv c_0 = \text{“t”} \land \text{even}(n) \land c_0' = \text{“e”} \land \text{UNCHANGED } \langle c_1, n \rangle \\
\mathit{Think}_0 &\equiv c_0 = \text{“e”} \land c_0' = \text{“t”} \land n' = n \,\text{div}\, 2 \land c_1' = c_1 \\
\mathit{Eat}_1 &\equiv c_1 = \text{“t”} \land \neg\text{even}(n) \land c_1' = \text{“e”} \land \text{UNCHANGED } \langle c_0, n \rangle \\
\mathit{Think}_1 &\equiv c_1 = \text{“e”} \land c_1' = \text{“t”} \land n' = 3 * n + 1 \land c_0' = c_0 \\
\mathit{Next} &\equiv \mathit{Eat}_0 \lor \mathit{Think}_0 \lor \mathit{Eat}_1 \lor \mathit{Think}_1 \\
\mathit{vars} &\equiv \langle c_0, c_1, n \rangle \\
\mathit{DM} &\equiv \mathit{Init} \land \Box[\mathit{Next}]_{\mathit{vars}} \land \text{WF}_{\mathit{vars}}(\mathit{Next})
\end{aligned}
$$

(b) TLA specification, concrete level

$$
\begin{aligned}
\overline{\mathit{Init}} &\equiv \overline{n} \in \{\mathbf{1}, \mathbf{2}\} \land \overline{c}_0 = \text{“t”} \land \overline{c}_1 = \text{“t”} \\
\overline{\mathit{Eat}_0} &\equiv \overline{c}_0 = \text{“t”} \land \overline{n} \in \{\mathbf{0}, \mathbf{2}\} \land \overline{c_0'} = \text{“e”} \land \text{UNCHANGED } \langle \overline{c}_1, \overline{n} \rangle \\
\overline{\mathit{Think}_0} &\equiv \land\, \overline{c}_0 = \text{“e”} \land \overline{c}_0' = \text{“t”} \land \overline{c}_1' = \overline{c}_1 \\
&\ \land \lor\, \overline{n} = \mathbf{0} \land \overline{n}' = \mathbf{0} \\
&\ \ \lor\, \overline{n} = \mathbf{1} \land \overline{n}' \in \{\mathbf{0}, \mathbf{1}, \mathbf{2}\} \\
&\ \ \lor\, \overline{n} = \mathbf{2} \land \overline{n'} \in \{\mathbf{1}, \mathbf{2}\} \\
\overline{\mathit{Eat}_1} &\equiv \overline{c}_1 = \text{“t”} \land \overline{n} = \mathbf{1} \land \overline{c}_1' = \text{“e”} \land \text{UNCHANGED } \langle \overline{c}_0, \overline{n} \rangle \\
\overline{\mathit{Think}_1} &\equiv \land\, \overline{c}_1 = \text{“e”} \land \overline{c}_1' = \text{“t”} \land \overline{c}_0' = \overline{c}_0 \\
&\ \land\, \overline{n}' = \textbf{if } \overline{n} \in \{\mathbf{0}, \mathbf{2}\} \textbf{ then 1 else 2} \\
\overline{\mathit{Next}} &\equiv \overline{\mathit{Eat}_0} \lor \overline{\mathit{Think}_0} \lor \overline{\mathit{Eat}_1} \lor \overline{\mathit{Think}_1} \\
\overline{\mathit{vars}} &\equiv \langle \overline{c}_0, \overline{c}_1, \overline{n} \rangle \\
\overline{\mathit{DM}} &\equiv \overline{\mathit{Init}} \land \Box[\overline{\mathit{Next}}]_{\overline{\mathit{vars}}} \land \text{WF}_{\overline{\mathit{vars}}}(\overline{\mathit{Next}})
\end{aligned}
$$

(c) TLA specification, abstract level

Figure 6.3: Dining mathematicians example.

only if n is even; it divides n by 2 when leaving the dining room. On the other hand, P_1 may eat only if n is odd and sets n to $3n+1$ when it resumes thinking. The left-hand side of figure 6.3(a) represents the states and transitions of the two processes.

We wish to verify the following properties of system DM:

$$\begin{array}{llll} (Pos) & \Box(n \in Nat \setminus \{0\}) & (Excl) & \Box\neg(c_0 = \text{“e”} \wedge c_1 = \text{“e”}) \\ (Live_0) & \Box\Diamond(c_0 = \text{“e”}) & (Live_1) & \Box\Diamond(c_1 = \text{“e”}) \end{array}$$

The invariance properties (Pos) and $(Excl)$ assert that the value of n is some positive natural number throughout any execution of DM and that the two processes never eat simultaneously. The liveness properties $(Live_0)$ and $(Live_1)$ assert freedom of starvation for both processes. Similarly as for the Bakery example, we attempt a proof by abstraction. The natural abstraction function suggested by the specification leaves the control variables unabstracted and replaces n by the abstract variable $\overline{n}$ that takes values in the set $\{\mathbf{0},\mathbf{1},\mathbf{2}\}$ where **0** represents the natural number 0 whereas **1** and **2** represent any odd or even positive natural number. Formally, the concrete and abstract variables are related by

$$\begin{array}{lll} hom & \equiv & \wedge\ \overline{c}_0 = c_0 \wedge \overline{c}_1 = c_1 \\ & & \wedge\ \overline{n} = \textbf{if } n = 0 \textbf{ then } \mathbf{0} \textbf{ else if } \text{even}(n) \textbf{ then } \mathbf{2} \textbf{ else } \mathbf{1} \end{array}$$

As the linking invariant we choose the transition formula

$$A \equiv n \in Nat \wedge [Next]_{vars}$$

Again, the invariant $DM \rightarrow \Box(n \in Nat)$ is easily established by static analysis.

Figure 6.3(c) shows a weakening $\overline{DM}$ of specification DM whose states and transitions (except for stuttering transitions) are illustrated in the right-hand side of figure 6.3(a). The correctness of the abstraction is established along similar lines as for the Bakery example; we have again used Isabelle to perform the necessary proofs. For example, the validity of

$$n \in Nat \wedge [Next]_{vars} \wedge hom \rightarrow (Think_0 \rightarrow \overline{Think_0})$$

relies on the fact that $(n \operatorname{div} 2)$ is non-zero if n is a positive and even natural number. Since $\overline{DM}$ describes a finite-state system, it can again be analysed using model checking. The following properties, which are of course strengthenings of the corresponding concrete properties, are easily read off the "abstract" state transition diagram of figure 6.3(a); their counterparts are therefore true for specification DM:

$$\begin{array}{llll} (\overline{Pos}) & \Box(\overline{n} \in \{\mathbf{1},\mathbf{2}\}) & (\overline{Excl}) & \Box\neg(\overline{c}_0 = \text{“e”} \wedge \overline{c}_1 = \text{“e”}) \\ (\overline{Live_0}) & \Box\Diamond(\overline{c}_0 = \text{“e”}) & & \end{array}$$

Unfortunately, we cannot verify $\overline{DM} \rightarrow \Box\Diamond(\overline{c}_1 = \text{“e”})$, which would imply starvation freedom for process P_1. In fact, the abstract system $\overline{DM}$ contains a cycle between the two states where $\overline{n}$ equals **2**, preventing P_1 from entering its “eating” state.

It is important to note that no finite-state homomorphic abstraction of specification DM, however fine-grained, allows to verify property $(Live_1)$. This is because for every $k \in \mathbb{N}$ there exists an execution of DM (choosing 2^k as the initial value of n) where process P_0 may eat k times in succession while P_1 has to wait. Any finite-state abstraction has to identify some states assumed during such an execution, for sufficiently large k, and must therefore contain a cycle of states where P_1 is not allowed to eat. On the other hand, the property $(Live_1)$ does hold for the concrete specification DM because P_0 divides n (which we know to be a positive natural number) by 2 whenever it executes $Think_0$, and therefore n eventually has to become odd. We will now see how to represent this idea within the framework of homomorphic abstractions.

Verification of liveness properties is in general based on the concept of a *well-founded relation* $(S, \prec)$, that is, an irreflexive relation $\prec \subseteq S \times S$ that does not admit infinite decreasing chains $s_0 \succ s_1 \succ \ldots$. Assuming a suitable first-order language with fixed interpretation of S and $\prec$, the TLA formula

$$\Box(t \in S) \rightarrow (\Box\Diamond\langle \mathbf{true} \rangle_t \rightarrow \Box\Diamond\langle \neg(t' \prec t) \rangle_t) \tag{6.6}$$

asserts that if the state function t always takes values from the set S and if infinitely often the value of t changes then there have to be infinitely many steps that change t without decreasing it. We can regard (6.6) as an alternative *definition* of the concept of a well-founded relation using temporal logic.

Theorem 6.7 *Assume that $\prec \subseteq S \times S$ is irreflexive. Formula (6.6) is valid iff $(S, \prec)$ is well-founded.*

Proof. “$\Rightarrow$”: If $(S, \prec)$ is not well-founded we may choose some decreasing sequence $c_0 \succ c_1 \succ \ldots$ with $c_i \in S$ for all i. Choose some behavior $\sigma = s_0 s_1 \ldots$ such that $s_i(x) = c_i$; this behavior satisfies $\Box(x \in S)$ as well as $\Box\Diamond\langle \mathbf{true} \rangle_x$ (because $\prec$ is irreflexive), but also $\Box[x' \prec x]_x$, and therefore formula (6.6) is not valid under this interpretation.

“$\Leftarrow$”: Assume that $(S, \prec)$ is well-founded and that behavior σ satisfies $\Box(t \in S)$ and $\Box\Diamond\langle \mathbf{true} \rangle_t$ as well as $\neg\Box\Diamond\langle \neg(t' \prec t) \rangle_t$, which is equivalent to $\Diamond\Box[t' \prec t]_t$, for some state function t. Therefore, there exists a suffix ρ of σ such that the values of t along ρ either decrease or remain unchanged with every step. The assumption $\sigma \models \Box\Diamond\langle \mathbf{true} \rangle_t$ implies that t changes value infinitely often along ρ, and therefore we obtain an infinite sequence of elements of S that decrease with respect to $\prec$, contradicting the assumption that $(S, \prec)$ is well-founded. Q.E.D.

From propositions 6.7 and 6.3 we obtain the instance

$$\frac{\begin{array}{c}\Phi \to \Box(t \in S) \\ \Phi^- \wedge \left(\Box\Diamond\langle \mathbf{true}\rangle_t^+ \to \Box\Diamond\langle\neg(t' \prec t)\rangle_t^-\right) \to P^+\end{array}}{\Phi \to P} \tag{6.7}$$

of rule (6.5) where $\langle \mathbf{true}\rangle_t^+$ and $\langle\neg(t' \prec t)\rangle_t^-$ are strengthening and weakening abstractions, respectively, of $\langle \mathbf{true}\rangle_t$ and $\langle\neg(t' \prec t)\rangle_t$ with respect to h and the strengthened linking invariant $A \wedge t \in S$..

The first hypothesis of rule (6.7) asserts an invariant of Φ that may be proved in the standard way, possibly even by "type checking", or by another proof by abstraction. The second hypothesis requires the abstract property to hold of the abstract system strengthened by the abstract counterpart of the well-foundedness principle. Observe that the second conjunct represents an additional (Streett-type) liveness hypothesis that is formulated at the abstract level and can therefore be useful for model checking.

Remark. Rule (6.7) leaves some room for choosing a strengthening abstraction of $\langle \mathbf{true}\rangle_t$: in principle, it is enough to pick some action that is guaranteed to change t and that can be recognized at the abstract level. On the other hand, for the liveness assumption added to Φ^- to be useful, it is advisable to choose a strengthening of (the disjunction of) those actions that are known to decrease t, because these will occur in any offending cycles reported by the model checker. Formally, we obviously have $\langle t' \prec t\rangle_t \to \langle \mathbf{true}\rangle_t$, and this justifies to replace rule (6.7) by

$$\frac{\begin{array}{c}\Phi \to \Box(t \in S) \\ \Phi^- \wedge \left(\Box\Diamond\langle t' \prec t\rangle_t^+ \to \Box\Diamond\langle\neg(t' \prec t)\rangle_t^-\right) \to P^+\end{array}}{\Phi \to P} \tag{6.8}$$

which, although weaker in theory, better indicates the intended application of (6.7).

Returning to the example of the "dining mathematicians", we choose the well-founded ordering $(\mathbb{N} \setminus \{0\}, <)$. We have already established the first hypothesis of rule (6.8) as the property (Pos). The only actions that change n are $Think_0$ and $Think_1$, and it is easy to see (and also easy to prove with Isabelle) that $\langle\overline{Think_0}\rangle_{\overline{vars}}$ and $\langle\overline{Think_1}\rangle_{\overline{vars}}$ are strengthening and weakening abstractions, respectively, of $\langle n' < n\rangle_n$ and $\langle\neg(n' < n)\rangle_n$ with respect to the abstraction function and the linking invariant $A \wedge (n \in Nat \setminus \{0\})$. Furthermore, an inspection of the abstract state transition diagram in figure 6.3(a) or a run of STeP's model checker confirms that

$$\overline{DM} \wedge \left(\Box\Diamond\langle\overline{Think}_0\rangle_{\overline{vars}} \rightarrow \Box\Diamond\langle\overline{Think}_1\rangle_{\overline{vars}}\right) \rightarrow \Box\Diamond(c_1 = \text{“e”})$$

because the cycle of figure 6.3(a) cannot be followed indefinitely due to the additional liveness assumption. Rule (6.8) allows us to conclude that DM implies property $(Live_1)$, and that the protocol is in fact starvation free for both processes.

6.6 Weaker fairness for better abstractions

We now address the second phenomenon mentioned in section 6.4 that may interfere with the verification of liveness properties over abstract models. From the definition of fairness properties and proposition 6.3 it follows that the weakening of a fairness condition $\mathrm{XF}_v(A)$, where XF is either WF or SF, requires a weakening abstraction of the transition formula $\langle A\rangle_v$, but a strengthening abstraction of its enabledness condition, ENABLED $\langle A\rangle_v$. It is quite likely that the abstraction function maps two concrete states s and t, only one of which enables the action, to the same abstract state $\overline{s}$. Because $\langle A\rangle_v$ is not enabled over both pre-images, the strengthening abstraction of the enabledness condition has to exclude $\overline{s}$. Ultimately, one may end up with **false** as the only possible strengthening, and the abstraction of the concrete fairness condition becomes trivial.

Rule (6.5) can be used to replace the original fairness condition at the concrete level by a weaker one whose enabledness is more faithfully represented at the abstract level. One way to weaken fairness conditions without detailed temporal reasoning about the concrete specification relies on the tautology

$$\mathrm{XF}_v(A) \wedge \mathrm{XF}_v(B) \rightarrow \mathrm{XF}_v(A \vee B)$$

An instance of this idea that we have found particularly useful occurs in the verification of parameterized systems that contain fairness conditions of the form $\mathrm{XF}_v(B(x))$ for each possible parameter value $x \in S$. Typically, one will try to establish properties of the form $\forall x \in S : P(x)$ that are also parametric. The first step in the proof introduces a fresh Skolem constant, say $\mathbf{x}$, reducing the proof to that of $\Phi \wedge \mathbf{x} \in S \rightarrow P(\mathbf{x})$, where Φ represents the specification of the system. In order to arrive at a finite-state abstraction of Φ, we will have to identify parameter values at the abstract level that are distinct at the concrete level. For the sake of the argument, suppose that in the abstract model we use only two parameters $\mathbf{x}$ and $\mathbf{y}$ where $\mathbf{y}$ represents any value different from $\mathbf{x}$. Any action $B(x)$ of the concrete system will be represented by two actions $\overline{B}_1(\mathbf{x})$ and $\overline{B}_2(\mathbf{y})$; the latter effectively representing the action $\exists y \in S \setminus \{\mathbf{x}\} : B(y)$ of the concrete system.

(See section 6.7 for an application of this technique.) This abstraction is clearly unable to faithfully represent the enabledness conditions of actions $B(y)$, for $y \in S \setminus \{\mathbf{x}\}$. Consequently, the fairness conditions associated with these actions will be trivialized in the abstraction $\overline{\Phi}$ of Φ, and liveness properties will be lost. In such cases we propose to apply the TLA rule

$$\frac{\begin{array}{l} A \wedge y \in M \wedge (\forall z \in M : [\neg B(z)]_v) \\ \quad \rightarrow (\text{ENABLED } \langle B(y)\rangle_v \rightarrow (\text{ENABLED } \langle B(y)\rangle_v)') \end{array}}{\Box A \wedge \forall y \in M : \mathrm{WF}_v(B(y)) \rightarrow \mathrm{WF}_v(\exists y \in M : B(y))} \quad M, y \text{ rigid} \tag{6.9}$$

that allows to replace universally quantified weak fairness conditions with a single fairness condition for a disjunction of parameterized actions. The hypothesis of rule (6.9) requires to prove that only actions $B(z)$, for some $z \in M$, disable $B(y)$; this ensures the existence of some value $y \in M$ such that action $B(y)$ remains persistently enables, provided the disjunction is persistently enabled.

Proposition 6.8 *Rule (6.9) is sound.*

Proof. Let σ be some behavior that satisfies both $\Box A \wedge \forall y \in M : \mathrm{WF}_v(B(y))$ and $\Diamond\Box\text{ENABLED } \langle \exists y \in M : B(y)\rangle_v$. Then there exists some suffix $\rho = r_0 r_1 \ldots$ of σ such that ENABLED $\langle \exists y \in M : B(y)\rangle_v$, and therefore $\exists y \in M :$ ENABLED $\langle B(y)\rangle_v$, holds of every state r_i (w.l.o.g., we may assume y not to be free in v). Choose some $\mathbf{y} \in M$ such that ENABLED $\langle B(\mathbf{y})\rangle_v$ holds of r_0. It suffices to prove that some step $\langle r_i, r_{i+1}\rangle$ satisfies $\exists y \in M : \langle B(y)\rangle_v$. Suppose not; we claim that ENABLED $\langle B(\mathbf{y})\rangle_v$ holds for all r_i. By the assumption that σ satisfies $\mathrm{WF}_v(B(\mathbf{y}))$ it follows that some step must satisfy $\langle B(\mathbf{y})\rangle_v$, and we have reached a contradiction.

We now prove the claim ENABLED $\langle B(\mathbf{y})\rangle_v$ for all r_i, by induction on i. For $i = 0$, the claim holds by assumption. For the inductive step, we note that $\langle r_i, r_{i+1}\rangle$ satisfies A and $\forall z \in M : [\neg B(z)]_v$, that $\mathbf{y} \in M$, and that r_i satisfies ENABLED $\langle B(\mathbf{y})\rangle_v$ (by induction hypothesis). Therefore we conclude that r_{i+1} satisfies ENABLED $\langle B(\mathbf{y})\rangle_v$ from the hypothesis of (6.9), which completes the proof. Q.E.D.

For strong fairness conditions we have the similar rule

$$\frac{\begin{array}{l} \Box A \wedge y \in M \wedge (\forall z \in M : \Box[\neg B(z)]_v) \\ \quad \rightarrow (\text{ENABLED } \langle B(y)\rangle_v \rightarrow \Box\Diamond\text{ENABLED } \langle B(y)\rangle_v) \end{array}}{\Box A \wedge \forall y \in M : \mathrm{SF}_v(B(y)) \rightarrow \mathrm{SF}_v(\exists y \in M : B(y))} \quad M, y \text{ rigid} \tag{6.10}$$

that is justified in a similar manner. The simpler rule

$$\frac{\mathit{finite}(M)}{\forall y \in M : \mathrm{XF}_v(B(y)) \rightarrow \mathrm{XF}_v(\exists y : B(y))} \quad M \text{ rigid} \tag{6.11}$$

is the straightforward generalization of the propositional rule mentioned at the beginning of this section to quantification over finite sets.

6.7 Verification of a reader-writer algorithm

As a complete example for the techniques that we have presented so far, we verify a reader-writer algorithm proposed by Hellwagner [44]. Its TLA specification is given as formula RW in figure 6.4.

The protocol is expressed in terms of five sets $actRd$, $waitRd$, $prioRd$, $actWr$, and $waitWr$ that contain (the identities of) currently active, waiting or prioritized readers or writers. All these sets are initially empty; the system can perform the following actions (for some process $p \in Proc$):

$ImmRd(p)$: Process p signals its desire to read when there are no active or waiting writers. It may immediately proceed to read the data.

$ReqRd(p)$: Process p requests reading access, but there are active or waiting writers. It is added to the set of waiting readers.

$DoRd(p)$: Prioritized reader p is allowed to read the data when there are no active writers.

$EndRd(p)$: Active reader p signals that it has finished reading and exits the protocol.

$ImmWr(p)$: Process p signals that it wants to write data when there are neither active or waiting writers nor active or prioritized readers. It may immediately proceed to write the data.

$ReqWr(p)$: Process p signals that it wants to write data, but the preconditions of action $ImmWr(p)$ are not met. It is added to the set of waiting writers.

$DoWr(p)$: Waiting writer p is allowed to write when there are neither active or prioritized readers nor active writers.

$EndWr(p)$: Active writer p signals that it has finished writing and exits the protocol. At the same time, all currently waiting readers are promoted to "prioritized readers".

The fairness conditions are chosen such that no process that has entered the system is persistently neglected. In particular, there is a strong fairness condition on the action $DoWr(p)$, for all $p \in Proc$, presumably implemented using a queue rather than a set of waiting writers.

$$
\begin{array}{rcl}
Init & \equiv & \land\ actRd = \emptyset \land waitRd = \emptyset \land prioRd = \emptyset \\
 & & \land\ actWr = \emptyset \land waitWr = \emptyset \\
ImmRd(p) & \equiv & \land\ actWr = \emptyset \land waitWr = \emptyset \land actRd' = actRd \cup \{p\} \\
 & & \land\ \text{UNCHANGED}\ \langle waitRd, prioRd, actWr, waitWr \rangle \\
ReqRd(p) & \equiv & \land\ \neg(actWr = \emptyset \land waitWr = \emptyset) \land waitRd' = waitRd \cup \{p\} \\
 & & \land\ \text{UNCHANGED}\ \langle actRd, prioRd, actWr, waitWr \rangle \\
DoRd(p) & \equiv & \land\ p \in prioRd \land actWr = \emptyset \\
 & & \land\ prioRd' = prioRd \setminus \{p\} \land actRd' = actRd \cup \{p\} \\
 & & \land\ \text{UNCHANGED}\ \langle waitRd, actWr, waitWr \rangle \\
EndRd(p) & \equiv & \land\ p \in actRd \land actRd' = actRd \setminus \{p\} \\
 & & \land\ \text{UNCHANGED}\ \langle waitRd, prioRd, actWr, waitWr \rangle \\
ImmWr(p) & \equiv & \land\ actRd = \emptyset \land prioRd = \emptyset \land actWr = \emptyset \land waitWr = \emptyset \\
 & & \land\ actWr' = actWr \cup \{p\} \\
 & & \land\ \text{UNCHANGED}\ \langle actRd, waitRd, prioRd, waitWr \rangle \\
ReqWr(p) & \equiv & \land\ \neg(actRd = \emptyset \land prioRd = \emptyset \land actWr = \emptyset \land waitWr = \emptyset) \\
 & & \land\ waitWr' = waitWr \cup \{p\} \\
 & & \land\ \text{UNCHANGED}\ \langle actRd, waitRd, prioRd, actWr \rangle \\
DoWr(p) & \equiv & \land\ p \in waitWr \land actRd = \emptyset \land prioRd = \emptyset \land actWr = \emptyset \\
 & & \land\ waitWr' = waitWr \setminus \{p\} \land actWr' = actWr \cup \{p\} \\
 & & \land\ \text{UNCHANGED}\ \langle actRd, waitRd, prioRd \rangle \\
EndWr(p) & \equiv & \land\ p \in actWr \land actWr' = actWr \setminus \{p\} \\
 & & \land\ prioRd' = prioRd \cup waitRd \land waitRd' = \emptyset \\
 & & \land\ \text{UNCHANGED}\ \langle actRd, waitWr \rangle \\
Next & \equiv & \exists p \in Proc : \lor\ ImmRd(p) \lor ReqRd(p) \lor DoRd(p) \lor EndRd(p) \\
 & & \qquad\qquad\quad \lor\ ImmWr(p) \lor ReqWr(p) \lor DoWr(p) \lor EndWr(p) \\
v & \equiv & \langle actRd, waitRd, prioRd, actWr, waitWr \rangle \\
RW & \equiv & \land\ Init \land \Box[Next]_v \\
 & & \land\ \forall p \in Proc : \land\ \mathrm{WF}_v(DoRd(p)) \land \mathrm{WF}_v(EndRd(p)) \\
 & & \qquad\qquad\qquad\ \land\ \mathrm{SF}_v(DoWr(p)) \land \mathrm{WF}_v(EndWr(p))
\end{array}
$$

Figure 6.4: Specification of the reader-writer algorithm

6.7.1 Mutual exclusion

Let us prove that the reader-writer algorithm ensures mutual exclusion between readers and writers as well as among writers, formally expressed as

$$
\begin{array}{ll}
(RWExcl) & \Box(actRd = \emptyset \lor actWr = \emptyset) \\
(WWExcl) & \Box(\forall p, q \in actWr : p = q)
\end{array}
$$

$$
\begin{array}{rcl}
\overline{Init} & \equiv & \land\ \overline{actRd} = 0 \land \overline{waitRd} = 0 \land \overline{prioRd} = 0 \\
 & & \land\ \overline{actWr} = 0 \land \overline{waitWr} = 0 \\
\overline{ImmRd} & \equiv & \land\ \overline{actWr} = 0 \land \overline{waitWr} = 0 \land \overline{actRd}' = 1 \\
 & & \land\ \text{UNCHANGED } \langle \overline{waitRd}, \overline{prioRd}, \overline{actWr}, \overline{waitWr} \rangle \\
\overline{ReqRd} & \equiv & \land\ \neg(\overline{actWr} = 0 \land \overline{waitWr} = 0) \land \overline{waitRd}' = 1 \\
 & & \land\ \text{UNCHANGED } \langle \overline{actRd}, \overline{prioRd}, \overline{actWr}, \overline{waitWr} \rangle \\
\overline{DoRd} & \equiv & \land\ \overline{prioRd} = 1 \land \overline{actWr} = 0 \land \overline{actRd}' = 1 \\
 & & \land\ \text{UNCHANGED } \langle \overline{waitRd}, \overline{actWr}, \overline{waitWr} \rangle \\
\overline{EndRd} & \equiv & \overline{actRd} = 1 \land \text{UNCHANGED } \langle \overline{waitRd}, \overline{prioRd}, \overline{actWr}, \overline{waitWr} \rangle \\
\overline{ImmWr} & \equiv & \land\ \overline{actRd} = 0 \land \overline{prioRd} = 0 \land \overline{actWr} = 0 \land \overline{waitWr} = 0 \\
 & & \land\ \overline{actWr}' = 1 \land \text{UNCHANGED } \langle \overline{actRd}, \overline{waitRd}, \overline{prioRd}, \overline{waitWr} \rangle \\
\overline{ReqWr} & \equiv & \land\ \neg(\overline{actRd} = 0 \land \overline{prioRd} = 0 \land \overline{actWr} = 0 \land \overline{waitWr} = 0) \\
 & & \land\ \overline{waitWr}' = 1 \land \text{UNCHANGED } \langle \overline{actRd}, \overline{waitRd}, \overline{prioRd}, \overline{actWr} \rangle \\
\overline{DoWr} & \equiv & \land\ \overline{waitWr} = 1 \land \overline{actRd} = 0 \land \overline{prioRd} = 0 \land \overline{actWr} = 0 \\
 & & \land\ \overline{actWr}' = 1 \land \text{UNCHANGED } \langle \overline{actRd}, \overline{waitRd}, \overline{prioRd} \rangle \\
\overline{EndWr} & \equiv & \land\ \overline{actWr} \neq 0 \land \overline{actWr}' \leq \overline{actWr} \\
 & & \land\ \overline{prioRd}' = \textbf{if } \overline{prioRd} = 0 \land \overline{waitRd} = 0 \textbf{ then } 0 \textbf{ else } 1 \\
 & & \land \land \overline{waitRd}' = 0 \land \text{UNCHANGED } \langle \overline{actRd}, \overline{waitWr} \rangle \\
\overline{Next} & \equiv & \lor\ \overline{ImmRd} \lor \overline{ReqRd} \lor \overline{DoRd} \lor \overline{EndRd} \\
 & & \lor\ \overline{ImmWr} \lor \overline{ReqWr} \lor \overline{DoWr} \lor \overline{EndWr} \\
\overline{v} & \equiv & \langle \overline{actRd}, \overline{waitRd}, \overline{prioRd}, \overline{actWr}, \overline{waitWr} \rangle \\
\overline{RW} & \equiv & \overline{Init} \land \Box[\overline{Next}]_{\overline{v}}
\end{array}
$$

Figure 6.5: Abstraction for mutual exclusion

We choose an abstraction function that can only distinguish between empty and non-empty sets, except for $actWr$, for which we use three possible values that correspond to empty, singleton or larger sets. Formally, the concrete and abstract levels are related by the state predicate

$$
\begin{array}{lll}
hom & \equiv & \land\ \overline{actRd} = \textbf{if } actRd = \emptyset \textbf{ then } 0 \textbf{ else } 1 \\
& & \land\ \overline{waitRd} = \textbf{if } waitRd = \emptyset \textbf{ then } 0 \textbf{ else } 1 \\
& & \land\ \overline{prioRd} = \textbf{if } prioRd = \emptyset \textbf{ then } 0 \textbf{ else } 1 \\
& & \land\ \overline{waitWr} = \textbf{if } waitWr = \emptyset \textbf{ then } 0 \textbf{ else } 1 \\
& & \land\ \overline{actWr} = \textbf{if } actWr = \emptyset \textbf{ then } 0 \textbf{ elsif } \forall p, q \in actWr : p = q \textbf{ then } 1 \textbf{ else } 2
\end{array}
$$

A weakening abstraction of specification RW with respect to this abstraction function and the linking invariant **true** is shown in figure 6.5; the correctness of the abstraction is easily established using proposition 6.3. Observe that the abstract specification is rather imprecise: for example, the $\overline{EndRd}$ action does not specify how the variable $\overline{actRd}$ changes. Besides, we have trivialized the liveness condition at the abstract level because the identity of individual processes has been lost in the abstraction. Analyzing specification $\overline{RW}$, STeP's model checker readily confirms the properties

$$
\begin{array}{ll}
(\overline{RWExcl}) & \Box(\overline{actRd} = 0 \lor \overline{actWr} = 0) \\
(\overline{WWExcl}) & \Box(\overline{actWr} \leq 1)
\end{array}
$$

which are of course strengthenings of the corresponding properties for the concrete specification. Comparing this abstraction-based proof with a traditional deductive verification of the invariants, we must admit that we have not really gained anything because the properties $(RWExcl)$ and $(WWExcl)$ are in fact inductive for specification RW. The effort needed for proving that $\overline{RW}$ is indeed a sound abstraction is therefore essentially the same as that for a direct proof of the invariants; as a matter of fact, both invariants can be proven with a single tactic invocation in the encoding of TLA in Isabelle/HOL [74].

6.7.2 Liveness properties

We now turn to the verification of liveness for each (reader or writer) process, expressed as

$$
\begin{array}{ll}
(RdLive) & \forall p \in Proc : p \in waitRd \leadsto p \in actRd \\
(WrLive) & \forall p \in Proc : p \in waitWr \leadsto p \in actWr
\end{array}
$$

As these are universally quantified formulas, we assume given some $\mathbf{p} \in Proc$ and prove

$$
\begin{array}{ll}
(RdLive_{\mathbf{p}}) & \mathbf{p} \in waitRd \leadsto \mathbf{p} \in actRd \\
(WrLive_{\mathbf{p}}) & \mathbf{p} \in waitWr \leadsto \mathbf{p} \in actWr
\end{array}
$$

$$
\begin{array}{rcl}
\overline{Init} & \equiv & \land\ \overline{actRd} = \emptyset \land \overline{waitRd} = \emptyset \land \overline{prioRd} = \emptyset \\
& & \land\ \overline{actWr} = \emptyset \land \overline{waitWr} = \emptyset \\
\overline{ImmRd}(p) & \equiv & \land\ \overline{actWr} = \emptyset \land \overline{waitWr} = \emptyset \land \overline{actRd}' = \overline{actRd} \cup \{p\} \\
& & \land\ \text{UNCHANGED } \langle \overline{waitRd}, \overline{prioRd}, \overline{actWr}, \overline{waitWr} \rangle \\
\overline{ReqRd}(p) & \equiv & \land\ \neg(\overline{actWr} = \emptyset \land \overline{waitWr} = \emptyset) \land \overline{waitRd}' = \overline{waitRd} \cup \{p\} \\
& & \land\ \text{UNCHANGED } \langle \overline{actRd}, \overline{prioRd}, \overline{actWr}, \overline{waitWr} \rangle \\
\overline{DoRd}(p) & \equiv & \land\ p \in \overline{prioRd} \land \overline{actWr} = \emptyset \\
& & \land\ \overline{actRd}' = \overline{actRd} \cup \{p\} \land \lor\ p = \mathbf{p} \land \overline{prioRd}' = \overline{prioRd} \setminus \{p\} \\
& & \qquad\qquad\qquad\qquad\qquad\quad \lor\ p \neq \mathbf{p} \land \overline{prioRd}' \subseteq \overline{prioRd} \\
& & \land\ \land \text{UNCHANGED } \langle \overline{waitRd}, \overline{actWr}, \overline{waitWr} \rangle \\
\overline{EndRd}(p) & \equiv & \land\ p \in \overline{actRd} \land \lor\ p = \mathbf{p} \land \overline{actRd}' = \overline{actRd} \setminus \{p\} \\
& & \qquad\qquad\qquad\quad \lor\ p \neq \mathbf{p} \land \overline{actRd}' \subseteq \overline{actRd} \\
& & \land\ \text{UNCHANGED } \langle \overline{waitRd}, \overline{prioRd}, \overline{actWr}, \overline{waitWr} \rangle \\
\overline{ImmWr}(p) & \equiv & \land\ \overline{actRd} = \emptyset \land \overline{prioRd} = \emptyset \land \overline{actWr} = \emptyset \land \overline{waitWr} = \emptyset \\
& & \land\ \overline{actWr}' = \overline{actWr} \cup \{p\} \\
& & \land\ \text{UNCHANGED } \langle \overline{actRd}, \overline{waitRd}, \overline{prioRd}, \overline{waitWr} \rangle \\
\overline{ReqWr}(p) & \equiv & \land\ \neg(\overline{actRd} = \emptyset \land \overline{prioRd} = \emptyset \land \overline{actWr} = \emptyset \land \overline{waitWr} = \emptyset) \\
& & \land\ \overline{waitWr}' = \overline{waitWr} \cup \{p\} \\
& & \land\ \text{UNCHANGED } \langle \overline{actRd}, \overline{waitRd}, \overline{prioRd}, \overline{actWr} \rangle \\
\overline{DoWr}(p) & \equiv & \land\ p \in \overline{waitWr} \land \overline{actRd} = \emptyset \land \overline{prioRd} = \emptyset \land \overline{actWr} = \emptyset \\
& & \land\ \overline{actWr}' = \overline{actWr} \cup \{p\} \land \lor\ p = \mathbf{p} \land \overline{waitWr}' = \overline{waitWr} \setminus \{p\} \\
& & \qquad\qquad\qquad\qquad\qquad\quad \lor\ p \neq \mathbf{p} \land \overline{waitWr}' \subseteq \overline{waitWr} \\
& & \land\ \text{UNCHANGED } \langle \overline{actRd}, \overline{waitRd}, \overline{prioRd} \rangle \\
\overline{EndWr}(p) & \equiv & \land\ p \in \overline{actWr} \land \lor\ p = \mathbf{p} \land \overline{actWr}' = \overline{actWr} \setminus \{p\} \\
& & \qquad\qquad\qquad\quad \lor\ p \neq \mathbf{p} \land \overline{actWr}' \subseteq \overline{actWr} \\
& & \land\ \overline{prioRd}' = \overline{prioRd} \cup \overline{waitRd} \land \overline{waitRd}' = \emptyset \\
& & \land\ \text{UNCHANGED } \langle \overline{actRd}, \overline{waitWr} \rangle \\
\overline{Next} & \equiv & \exists p \in \{\mathbf{p}, \mathbf{q}\} : \lor\ \overline{ImmRd}(p) \lor \overline{ReqRd}(p) \lor \overline{DoRd}(p) \lor \overline{EndRd}(p) \\
& & \qquad\qquad\quad\ \ \lor\ \overline{ImmWr}(p) \lor \overline{ReqWr}(p) \lor \overline{DoWr}(p) \lor \overline{EndWr}(p) \\
\overline{v} & \equiv & \langle \overline{actRd}, \overline{waitRd}, \overline{prioRd}, \overline{actWr}, \overline{waitWr}, \overline{t} \rangle \\
\overline{RW} & \equiv & \land\ \overline{Init} \land \overline{t} = 0 \land \Box[\overline{Next} \land \overline{t}' = 1 - \overline{t}]_{\overline{v}} \\
& & \land\ \forall p \in \{\mathbf{p}, \mathbf{q}\} : \mathrm{WF}_{\overline{v}}(\overline{DoRd}(p)) \land \mathrm{WF}_{\overline{v}}(\overline{EndRd}(p)) \land \mathrm{WF}_{\overline{v}}(\overline{EndWr}(p)) \\
& & \land\ \mathrm{SF}_{\overline{v}}(\overline{DoWr}(\mathbf{p})) \land \mathrm{WF}_{\overline{v}}(\overline{DoWr}(\mathbf{q}))
\end{array}
$$

Figure 6.6: Abstraction for liveness

The abstraction must now be able to keep track of process **p**'s way through the protocol. On the other hand, all processes $q \in \mathit{Proc} \setminus \{\mathbf{p}\}$ are mapped to the same value **q** at the abstract level. Formally we use the abstraction function

$$
\begin{aligned}
h(S) \quad = \quad & \textbf{if } S = \emptyset \textbf{ then } \emptyset \\
& \textbf{elsif } S = \{\mathbf{p}\} \textbf{ then } \{\mathbf{p}\} \\
& \textbf{elsif } \mathbf{p} \in S \textbf{ then } \{\mathbf{p}, \mathbf{q}\} \\
& \textbf{else } \{\mathbf{q}\}
\end{aligned}
$$

to relate the variables at the abstract and concrete levels. This abstraction does not ensure that transitions that change the variables at the concrete level are mapped into visible transitions at the abstract level, which is required for the abstraction of fairness conditions. We therefore first augment specification RW with a history variable t whose value changes with every transition. By rule (∃-H) the augmented specification is equivalent to the original one. We leave variable t unabstracted in the weakening abstraction $\overline{RW}$ shown in figure 6.6. The idea behind this abstraction is to consider separately the effect of each action on process **p** and on processes different from **p**. The correctness of the action weakenings w.r.t. the linking invariant

$$A \quad \equiv \quad t \in \{0,1\} \wedge [\mathit{Next} \wedge t' = 1 - t]_{v,t}$$

that corresponds to the augmented specification requires little justification and is readily shown using Isabelle.

Concerning the fairness conditions, we similarly distinguish between actions concerning **p** and those concerning some other process. Formally, we equivalently rewrite the fairness conditions of the concrete-level specification RW by

$$
\begin{aligned}
& \wedge\ \mathrm{WF}_v(\mathit{DoRd}(\mathbf{p})) \wedge \forall q \in \mathit{Proc} \setminus \{\mathbf{p}\} : \mathit{WF}_v(\mathit{DoRd}(q)) \\
& \wedge\ \mathrm{WF}_v(\mathit{EndRd}(\mathbf{p})) \wedge \forall q \in \mathit{Proc} \setminus \{\mathbf{p}\} : \mathit{WF}_v(\mathit{EndRd}(q)) \\
& \wedge\ \mathrm{SF}_v(\mathit{DoWr}(\mathbf{p})) \wedge \forall q \in \mathit{Proc} \setminus \{\mathbf{p}\} : \mathit{SF}_v(\mathit{DoWr}(q)) \\
& \wedge\ \mathrm{WF}_v(\mathit{EndWr}(\mathbf{p})) \wedge \forall q \in \mathit{Proc} \setminus \{\mathbf{p}\} : \mathit{WF}_v(\mathit{EndWr}(q))
\end{aligned}
$$

The abstraction function faithfully maps enabledness conditions for the actions concerning **p** to the enabledness conditions for their counterparts at the abstract level. This justifies to add the conditions

$$
\begin{aligned}
& \wedge\ \mathrm{WF}_{\overline{v}}(\overline{\mathit{DoRd}}(\mathbf{p})) \wedge \mathrm{WF}_{\overline{v}}(\overline{\mathit{EndRd}}(\mathbf{p})) \\
& \wedge\ \mathrm{SF}_{\overline{v}}(\overline{\mathit{DoWr}}(\mathbf{p})) \wedge \mathrm{WF}_{\overline{v}}(\overline{\mathit{EndWr}}(\mathbf{p}))
\end{aligned}
$$

to the specification $\overline{RW}$. For the remaining conditions we intend to apply rule (6.9). To simplify matters, we first replace the strong fairness condition on the

$DoWr$ action by weak fairness; this can be considered as yet another (trivial) instance of rule (6.5). It is easy to prove the hypothesis of rule (6.9) for each of the actions. After this quantifier shift the enabledness conditions and effects of the action formulas at the concrete and abstract levels match as required (it is here that we use the strengthened linking invariant that implies that $\overline{v}$ changes with every action), and we arrive at the fairness conditions shown in figure 6.6.

We can now try to establish the properties

$$\begin{array}{ll} (\overline{RdLive}) & \mathbf{p} \in \overline{waitRd} \leadsto \mathbf{p} \in \overline{actRd} \\ (\overline{WrLive}) & \mathbf{p} \in \overline{waitWr} \leadsto \mathbf{p} \in \overline{actWr} \end{array}$$

of specification $\overline{RW}$ using STeP's model checker. Unfortunately, it produces a counter-example that ends in a cycle of $EndRd(\mathbf{q})$ steps without $\overline{actRd}$ ever becoming empty. Obviously, such a run does not have a counterpart in the concrete specification because all sets are finite during any run of the system. Clearly, this is an instance of spurious cycles in the abstract system, which can be broken using well-founded orderings and rule (6.8). It turns out that we have to consider the sets $actRd$, $actWr$, and $prioRd$ and identify the actions that either add or remove elements. Omitting the details, which are similar to the "dining mathematicians" specification considered in section 6.5, we find that it suffices to verify the property

$$\begin{array}{ll} & \Box\Diamond\langle \overline{EndRd}(\mathbf{p}) \vee \overline{EndRd}(\mathbf{q})\rangle_{\overline{v}} \\ & \quad \rightarrow \Box\Diamond\langle \overline{ImmRd}(\mathbf{p}) \vee \overline{ImmRd}(\mathbf{q}) \vee \overline{DoRd}(\mathbf{p}) \vee \overline{DoRd}(\mathbf{q})\rangle_{\overline{v}} \\ \wedge & \Box\Diamond\langle \overline{EndWr}(\mathbf{p}) \vee \overline{EndWr}(\mathbf{q})\rangle_{\overline{v}} \\ & \quad \rightarrow \Box\Diamond\langle \overline{ImmWr}(\mathbf{p}) \vee \overline{ImmWr}(\mathbf{q}) \vee \overline{DoWr}(\mathbf{p}) \vee \overline{DoWr}(\mathbf{q})\rangle_{\overline{v}} \\ \wedge & \Box\Diamond\langle \overline{DoRd}(\mathbf{p}) \vee \overline{DoRd}(\mathbf{q})\rangle_{\overline{v}} \\ & \quad \rightarrow \Box\Diamond\langle \overline{EndWr}(\mathbf{p}) \vee \overline{EndWr}(\mathbf{q})\rangle_{\overline{v}} \\ \rightarrow & (\mathbf{p} \in \overline{waitRd} \leadsto \mathbf{p} \in \overline{actRd}) \end{array}$$

in order to establish $(RdLive)$ for the concrete specification RW, and similarly for the property $(WrLive)$. Both are readily confirmed by STeP's model checker.

As in the case of the Bakery algorithm (cf. section 6.3), we observe that the abstraction techniques are particularly helpful for proving liveness properties of the reader-writer algorithm. In fact, traditional deductive verification would have required numerous applications of rules based on the fairness of helpful transitions as well as the ordering arguments implicit in the above discussion. Most of the tedium involved in these proofs can be left to the model checker. Methodologically, we suggest the following approach:

- Start with a simple abstraction function that keeps the abstract state space as small as possible.
- Try to establish the desired safety properties. If necessary, refine the abstraction function, guided by the counter examples produced by the model checker.
- For the verification of liveness properties, start out with the same abstract model as for the safety properties. Add some simple fairness conditions that are clearly abstractions of those at the concrete level. Depending on the results of model checking, refine the abstract model by using a finer abstraction function or by applying the rules of sections 6.5 and 6.6 to eliminate counter examples that have no counterpart over the concrete model.

Chapter 7

Diagrammatic verification

7.1 Visualizing abstractions

In chapter 6 we have studied the technique of homomorphic abstractions as a basis for integrating interactive and automatic verification. We have shown that the choice of suitable abstractions, and in particular the judicious use of techniques to strengthen homomorphic abstractions can significantly reduce the number of proof obligations that need to be proven interactively, particularly when verifying liveness properties.

However, in that approach the user is still required to give an explicit specification of the abstract model and to verify the correctness of the abstract specification with respect to the abstraction function and a suitable linking invariant. Often, a better alternative is to describe the abstract model, which we expect to be finite, in terms of a state-transition diagram. The nodes are labelled with predicates over the state space of the concrete model, edge annotations indicate which actions are responsible for the transition and possibly their effect on terms over well-founded orderings. Although theoretically equivalent, such a format for representing abstractions by predicates on the concrete state space is attractive from a practical point of view: first, abstractions can be presented in an intuitive way and second, no explicit linking invariant is required because the predicates that appear in the node labels can be directly interpreted on the concrete state space.

We describe the proof obligations that ensure that such a state-transition diagram corresponds to (i.e., weakens) a specification. We also study techniques for the generation of diagrams for a given specification and a set of predicates of interest. In some cases, the model checker can establish properties directly from the generated diagrams, resulting in a technique for the automatic verification of infinite-state systems. When this is impossible, diagrams can be refined interactively, guided by the counter examples the failed verification attempt produces.

We also define formal refinement notions at the level of diagrams that can support a top-down style of system development.

7.2 Predicate diagrams

Assume a language with a designated (finite) set $\mathcal{O}$ of binary relation symbols $\prec$, written infix, that are interpreted by well-founded orderings over suitable sets. We assume that $x \prec y$ is true only if both x and y are elements of the carrier set of the ordering. We denote by $\preceq$ the reflexive closure of $\prec$, and write $\mathcal{O}^=$ to denote the set of relation symbols $\prec$ and $\preceq$, for $\prec \in \mathcal{O}$. By abuse of notation, we also write $\prec$ and $\preceq$ to denote the interpretations of the symbols in $\mathcal{O}^=$.

A *predicate diagram* is a finite graph whose nodes are labelled with sets of (possibly negated) predicates, and whose edges are labelled with actions (more precisely, action names) as well as optional annotations that assert terms to decrease with respect to an ordering in $\mathcal{O}^=$. Intuitively, a node of a predicate diagram represents the set of system states that satisfy the formulas contained in the node. (We indifferently write n for the set and the conjunction of its elements.) An edge (n,m) is labelled with action A if the action may cause a transition from a state represented by n to a state represented by m. An action A may have an associated fairness condition; fairness conditions apply to all transitions labelled by the action rather than to individual edges. We let edges be labelled with action names instead of action formulas because, in a top-down development, the precise action formula that defines an action is not known until the final specification has been derived.

Formally, the definition of predicate diagrams is relative to finite sets $\mathcal{P}$ and $\mathcal{A}$ that contain the state predicates and the (names of) actions of interest; we will later use $\tau \notin \mathcal{A}$ to denote a special stuttering action. We write $\overline{\mathcal{P}}$ to denote the set of literals formed from the predicates in $\mathcal{P}$, that is, the union of $\mathcal{P}$ and the negations of the predicates in $\mathcal{P}$.

Definition 7.1 *Assume given finite sets $\mathcal{P}$ and $\mathcal{A}$ of state predicates and action names. A* predicate diagram $G = (N, I, \delta, o, \zeta)$ *over $\mathcal{P}$ and $\mathcal{A}$ consists of*

- *a finite set $N \subseteq 2^{\overline{\mathcal{P}}}$ of* nodes,
- *a finite set $I \subseteq N$ of* initial nodes,
- *a family $\delta = (\delta_A)_{A \in \mathcal{A}}$ of relations $\delta_A \subseteq N \times N$; we also denote by δ the union of the relations δ_A, for $A \in \mathcal{A}$ and write $\delta_=$ to denote the reflexive closure of the union of these relations,*

- *an edge labelling o that associates a finite set $\{(t_1, \prec_1), \ldots, (t_k, \prec_k)\}$ of terms t_i paired with a relation $\prec_i \in O^=$ with every edge $(n, m) \in \delta$, and*

- *a mapping $\zeta : \mathcal{A} \to \{\mathrm{NF}, \mathrm{WF}, \mathrm{SF}\}$ that associates a fairness condition with every action in $\mathcal{A}$; the possible values represent no fairness, weak fairness, and strong fairness.*

We say that the action $A \in \mathcal{A}$ can be taken *at node $n \in N$ iff $(n, m) \in \delta_A$ holds for some $m \in N$. We write En(A) to denote the set of nodes $n \in N$ where A can be taken.*

We now define runs and traces through a diagram as the set of those behaviors that correspond to fair runs satisfying the node and edge labels. To evaluate the fairness conditions, we identify the enabling condition of an action $A \in \mathcal{A}$ with the existence of A-labelled edges at a given node.

Definition 7.2 *Let $G = (N, I, \delta, o, \zeta)$ be a predicate diagram over sets $\mathcal{P}$ and $\mathcal{A}$ of state predicates and actions and α be a valuation of the rigid variables. A* run *of G (w.r.t. α) is an ω-sequence $\rho = (s_0, n_0, A_0)(s_1, n_1, A_1)\ldots$ of triples where s_i is a state, $n_i \in N$ is a node, and $A_i \in \mathcal{A} \cup \{\tau\}$ is an action such that all of the following conditions hold:*

- *$n_0 \in I$ is an initial node,*

- *$s_i[\![n_i]\!]$ holds for all $i \in \mathbb{N}$,*

- *for all $i \in \mathbb{N}$, either $A_i = \tau$ and $n_i = n_{i+1}$, or $A_i \in \mathcal{A}$ and $(n_i, n_{i+1}) \in \delta_{A_i}$,*

- *if $A_i \in \mathcal{A}$ and $(t, \prec) \in o(n_i, n_{i+1})$, then $s_{i+1}[\![t]\!] \prec s_i[\![t]\!]$,*

- *if $A_i = \tau$ then $s_{i+1}[\![t]\!] \preceq s_i[\![t]\!]$ holds whenever $(t, \prec) \in o(n_i, m)$ for some $m \in N$,*

- *for every action $A \in \mathcal{A}$ such that $\zeta(A) = \mathrm{WF}$ there are infinitely many $i \in \mathbb{N}$ such that either $A_i = A$ or $n_i \notin$ En(A), and*

- *for every action $A \in \mathcal{A}$ such that $\zeta(A) = \mathrm{SF}$, either $A_i = A$ holds for infinitely many $i \in \mathbb{N}$ or $n_i \in$ En(A) holds for only finitely many $i \in \mathbb{N}$.*

We write runs(G) to denote the set of runs of G.

The set tr(G) of traces through *G consists of all behaviors $\sigma = s_0 s_1 \ldots$ such that there exists a run $\rho = (s_0, n_0, A_0)(s_1, n_1, A_1)\ldots$ of G based on the states in σ.*

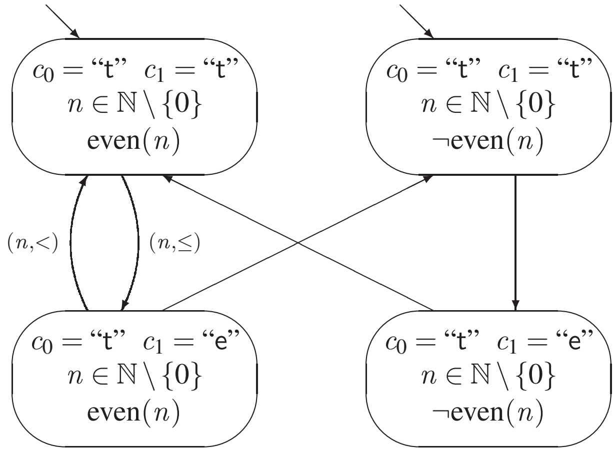


Figure 7.1: Predicate diagram for the "dining mathematicians" example.

Informally, $\sigma = \langle s_0, s_1, \ldots \rangle$ is a trace through the predicate diagram G if we can find a sequence of nodes n_i whose associated formulas are true at s_i and that are related by transitions whose edge labels, including the ordering annotations, are satisfied by consecutive states. In addition to the transitions that are explicitly represented by edges of the diagram, we allow stuttering transitions that remain in the source node. This definition ensures that the set of traces through a diagram is closed under stuttering equivalence, just as the models of TLA* formulas. Note that we do not require that states s_i and s_{i+1} related by a stuttering step (i.e., $A_i = \tau$) be identical; they are only required to satisfy the same node label. This interpretation allows such transitions to be refined later on, as we will see in section 7.5. However, if node n_i has an outgoing edge with an ordering annotation $(t, \prec)$, stuttering transitions are forbidden to increase the value of t, as otherwise stuttering could interfere with liveness properties induced by well-founded orderings.

Fairness conditions are used to prevent infinite stuttering. Their interpretation is standard, based on the intuition that the enabledness of actions with non-trivial fairness requirements is reflected in the diagram.

For example, let us reconsider the "dining mathematicians" specification studied in section 6.5. A predicate diagram based on this specification is shown in figure 7.1. That figure is based on the following conventions that we will continue to use when presenting predicate diagrams: the sets $\mathcal{P}$ and $\mathcal{A}$ contain the predicates and action names that appear in the diagram; moreover, the action name $Next$ is omitted.

Traces through the diagram are ω-sequences of states that satisfy the node labels and are related by edges shown in the diagram or by stuttering transitions. For example, the diagram rules out transitions from a state represented by the upper left node to one represented by the lower right node. In addition, the ordering annotations exclude behaviors that cycle forever between the two left-hand nodes because the value assigned to the variable n would have to decrease infinitely often, but can never increase—not even during stuttering transitions, because both nodes have outgoing edges labelled by $(n, <)$ or $(n, \leq)$. Assuming weak fairness for $Next$, it follows that all states of the diagram must be visited infinitely often during each run.

7.3 System verification via predicate diagrams

We now describe the use of predicate diagrams in order to verify TLA specifications of reactive systems. In linear-time formalisms such as TLA, trace inclusion is the appropriate implementation relation. Thus, a specification $Spec$ implements a property or high-level specification F if and only if the implication $Spec \rightarrow F$ is valid [62]. Predicate diagrams can be used to reduce the proof of this implication to two conditions: first, all behaviors allowed by $Spec$ must also be traces through the diagram (in other words, the diagram is a correct abstraction of $Spec$), and second, every trace through the diagram must satisfy F. Although both conditions are stated in terms of trace inclusion, we use different proof techniques in order to automate the verification process as much as possible. When establishing the correctness of the abstraction, we consider the node and edge labels of diagrams as predicates on the concrete state space of $Spec$, and reduce trace inclusion to a set of first-order verification conditions that concern individual states and transitions. On the other hand, to show that the diagram implies the high-level property, we regard all labels as Boolean variables. The predicate diagram can therefore be encoded as a finite labelled transition system, whose temporal properties can be established by model checking. In this respect, predicate diagrams act as an interface between interactive and automatic proof methods. We now consider both aspects in more detail.

7.3.1 Relating specifications and predicate diagrams

When comparing a TLA specification and a predicate diagram, we must first assign meaning to the action names that appear in the diagram. We assume given a function α that assigns an action formula to every action name. Because no confusion is possible, we will leave this assignment implicit, and again write A instead of $\alpha(A)$ when referring to the formula assigned to the name A.

We say that a predicate diagram G *conforms* to a specification $Spec$ if every behavior that satisfies $Spec$ is a trace through G. In general, proving conformance requires reasoning about entire behaviors. The following theorem essentially introduces a set of first-order ("local") verification conditions that are sufficient to establish conformance of a diagram to a system specification in standard form.[1] These conditions can either be checked by an (interactive or automatic) theorem prover or they can be ensured during the construction of a diagram from a given specification.

Theorem 7.3 *Let $G = (N, I, \delta, o, \zeta)$ be a predicate diagram over $\mathcal{P}$ and $\mathcal{A}$, and $\mathit{Spec} \equiv \mathit{Init} \wedge \Box[\mathit{Next}]_v \wedge L$ be a system specification. If all of the following conditions hold then G conforms to Spec.*

1. $\models \mathit{Init} \rightarrow \bigvee_{n \in I} n$

2. $\models n \wedge [\mathit{Next}]_v \rightarrow n' \vee \bigvee_{\{(A,m):(n,m)\in\delta_A\}} \langle A \rangle_v \wedge m'$ *holds for every node* $n \in N$.

3. *For all* $n, m \in N$ *and all* $(t, \prec) \in o(n, m)$*:*

   (a) $\models n \wedge m' \wedge \bigvee_{\{A:(n,m)\in\delta_A\}} \langle A \rangle_v \rightarrow t' \prec t$ *and*

   (b) $\models n \wedge [\mathit{Next}]_v \wedge n' \rightarrow t' \preceq t$.

4. *For every action* $A \in \mathcal{A}$ *such that* $\zeta(A) \neq \mathrm{NF}$*:*

   (a) *If* $\zeta(A) = \mathrm{WF}$ *then* $\models \mathit{Spec} \rightarrow \mathrm{WF}_v(A)$.

   (b) *If* $\zeta(A) = \mathrm{SF}$ *then* $\models \mathit{Spec} \rightarrow \mathrm{SF}_v(A)$.

   (c) $\models n \rightarrow \mathrm{ENABLED}\ \langle A \rangle_v$ *holds whenever* $n \in \mathit{En}(A)$.

   (d) $\models n \wedge \langle A \rangle_v \rightarrow \neg m'$ *holds for all* $n, m \in N$ *such that* $(n, m) \notin \delta_A$.

Proof. Assume that Spec and G are such that all conditions hold, that α is a valuation and that $\sigma = s_0 s_1 \ldots$ that σ is a behavior such that $\sigma \models \mathit{Spec}$. We inductively define a sequence $n_0 n_1 \ldots$ of nodes $n_i \in N$ and a sequence $\mathcal{A}_0 \mathcal{A}_1 \ldots$ of sets $\emptyset \neq \mathcal{A}_i \subseteq \mathcal{A} \cup \{\tau\}$ such that for all $i \in \mathbb{N}$ the following conditions hold:

(i) $s_i[\![n_i]\!]$,

(ii) $n_0 \in I$,

(iii) $\mathcal{A}_i = \{A \in \mathcal{A} : (n_i, n_{i+1}) \in \delta_A$ and $s_i[\![\langle A \rangle_v]\!]s_{i+1}\} \cup \{\tau : n_{i+1} = n_i\}$,

(iv) $s_{i+1}[\![t]\!] \prec s_i[\![t]\!]$ whenever $(t, \prec) \in o(n_i, n_{i+1})$, and

(v) $s_{i+1}[\![t]\!] \preceq s_i[\![t]\!]$ whenever $n_{i+1} = n_i$ and $(t, \prec) \in o(n_i, m)$ for some $m \in N$.

[1] The generalization to system specifications with several conjuncts $\Box[N_i]_{v_i}$ is straightforward.

Obviously, for any choice of $A_i \in \mathcal{A}_i$, conditions (i)–(v) ensure the conditions given in definition 7.2 for establishing $\sigma \in tr(G)$, except for the conditions related to fairness requirements.

For the induction base, we choose some node $n_0 \in I$ such that $s_0[\![n_0]\!]$ holds; the existence of some such node is ensured by condition (7.3.1), since $s_0[\![Init]\!]$ holds by assumption.

For the induction step, we assume that $n_0, \ldots, n_i$ and $\mathcal{A}_0, \ldots, \mathcal{A}_{i-1}$ have already been defined such that conditions (i) and (ii) holds for all $j \leq i$ and that conditions (iii)–(v) hold for all $j \leq i-1$. In particular, we have $s_i[\![n_i]\!]$. Moreover, the assumption that $\sigma \models Spec$ implies that $s_i[\![[Next]_v]\!]s_{i+1}$, and condition (7.3.2) ensures that either there exist some action $A \in \mathcal{A}$ and some $m \in N$ such that $(n_i, m) \in \delta_A$ and $s_i[\![\langle A\rangle_v \wedge m']\!]s_{i+1}$ holds, or $s_{i+1}[\![n_i]\!]$. In the first case, choose some such node m as n_{i+1}; in the second case, choose $n_{i+1} = n_i$. In either case, define $\mathcal{A}_i$ as described in condition (iii). These choices imply that $s_{i+1}[\![n_{i+1}]\!]$ and that $\mathcal{A}_i \neq \emptyset$. Conditions (iv) and (v) now follow from the choices of n_{i+1} and $\mathcal{A}_i$ with the help of conditions (7.3.3a) and (7.3.3b).

To complete the proof of the theorem, it remains to pick a sequence $A_0 A_1 \ldots$ of actions such that the last two conditions of definition 7.2 that concern the fairness annotations in G are satisfied. Choose a sequence such that $A_i = A$ holds for infinitely many $i \in \mathbb{N}$ if $A_i \in \mathcal{A}_i$ holds for infinitely many $i \in \mathbb{N}$. Such a choice is possible by a standard combinatorial argument since the set $\mathcal{A}$ is finite.

Now assume that $A \in \mathcal{A}$ is an action such that $\zeta(A) = \mathrm{WF}$ and that $n_i \in En(A)$ holds for all but finitely many $i \in \mathbb{N}$ (otherwise the condition is trivially satisfied). Because we already know that $s_i[\![n_i]\!]$ holds for all $i \in \mathbb{N}$, condition (7.3.4c) implies that $s_i[\![\text{ENABLED } \langle A\rangle_v]\!]$ holds for all but finitely many $i \in \mathbb{N}$. Moreover, since σ satisfies $Spec$ and $\models Spec \rightarrow \mathrm{WF}_v(A)$ by condition (7.3.4a), it follows that $s_i[\![\langle A\rangle_v]\!]s_{i+1}$ holds for infinitely many $i \in \mathbb{N}$. For every such i, we have $A \in \mathcal{A}_i$: otherwise, condition (7.3.4d) would imply that $s_{i+1}[\![\neg n_{i+1}]\!]$, contradicting assertion (i) above. But then, the choice of the action sequence implies that $A_i = A$ holds for infinitely many $i \in \mathbb{N}$, which completes the proof.

For actions $A \in \mathcal{A}$ such that $\zeta(A) = \mathrm{SF}$, the proof is analogous, replacing "all but finitely many" by "infinitely many" and using (7.3.4b) instead of (7.3.4a). Q.E.D.

Theorem 7.3 can be used to show that the predicate diagram of figure 7.1 conforms to the specification DM shown in figure 6.3. For example, we have

$$\begin{array}{l}\wedge\ c_0 = \text{“e”} \wedge c_1 = \text{“t”} \wedge n \in Nat \wedge n \neq 0 \wedge \mathrm{even}(n) \\ \wedge\ c_0' = \text{“t”} \wedge c_1' = \text{“t”} \wedge n' \in Nat \wedge n' \neq 0 \wedge \mathrm{even}(n') \\ \wedge\ \langle Next\rangle_v \\ \rightarrow\ n' < n\end{array}$$

because only the subaction Thk_0 is applicable in any state described by the precondition, and $n \operatorname{div} 2 < n$ holds for any positive integer n.

Except for the two conditions (7.3.4a) and (7.3.4b) that assert that the fairness conditions associated with the diagram are implied by the specification, all proof obligations stated in theorem 7.3 are non-temporal and can in principle be discharged using conventional theorem provers. We expect that conditions (7.3.4a) and (7.3.4b) are trivial to establish whenever theorem 7.3 is applied in that these fairness conditions appear as conjuncts of the specification's liveness condition L. Section 7.5 will introduce techniques that can be used to refine high-level fairness conditions before applying theorem 7.3. Nevertheless, observe that the number of verification conditions that have to be checked to prove conformance can be quadratic in the number of nodes in the diagram, due to condition (7.3.4d). In section 7.4 we will present techniques based on abstract interpretation that can construct predicate diagrams that conform to a given specification, obviating the need for explicit proofs.

7.3.2 Model checking predicate diagrams

For the proof that all traces through a predicate diagram satisfy some property F we view the predicate diagram as a finite transition system that is amenable to model checking. All predicates and actions that appear as labels of nodes or edges are then viewed as atomic propositions.

Formally, assume given a predicate diagram $G = (N, I, \delta, \pi, o, \zeta)$ over $\mathcal{P}$ and $\mathcal{A}$. We introduce new boolean variables b_P for every predicate $P \in \mathcal{P}$. A variable $action$ takes values in $\mathcal{A} \cup \{\tau\}$ and is used to indicate the last action taken. We also add variables $b_{(t,\prec)}$, for every term t and relation $\prec \in O$ such that $(t, \prec)$ appears in some ordering annotation $o(n,m)$ of G. We let $\overline{b}$ denote the set that contains the variables b_P, $action$, and $b_{(t,\prec)}$. We write $\overline{\varphi}$ for the formula that results from φ by replacing all occurrences of $P \in \mathcal{P}$ by b_P. The *characteristic formula* χ_G for diagram G is built from the variables in $\overline{b}$ and the auxiliary variable c that changes value whenever a step is taken. We introduce the following abbreviations:

- By $\mathcal{T}$ we denote the set of pairs $(t, \prec)$ where $\prec \in O$ that appear in the annotation of some transition in G. For $(t, \prec) \in \mathcal{T}$ we define the sets

$$
\begin{array}{rcl}
dec(t,\prec) & = & \{(n,m) : (t,\prec) \in o(n,m)\} \\
ninc(t,\prec) & = & \{(n,m) : (t,\preceq) \in o(n,m) \text{ or} \\
 & & \qquad (n = m \text{ and } (t,\prec) \in o(n,q) \text{ or } (t,\preceq) \in o(n,q) \\
 & & \qquad\qquad \text{for some } q \in N)\} \\
unk(t,\prec) & = & (N \times N) \setminus (dec(t,\prec) \cup ninc(t,\prec))
\end{array}
$$

- For an edge $(n,m) \in \delta$ we define the action formula

$$\begin{array}{rcl} taken(n,m) & \equiv & \wedge\ \overline{n} \wedge \overline{m}' \\ & & \wedge \displaystyle\bigvee_{\{A:(n,m)\in\delta_A\}} action' = A \ \vee \bigvee_{n=m} action' = \tau \\ & & \wedge \displaystyle\bigwedge_{(n,m)\in dec(t,\prec)} b_{(t,\prec)} = 2 \\ & & \wedge \displaystyle\bigwedge_{(n,m)\in ninc(t,\prec)} b_{(t,\prec)} = 1 \\ & & \wedge \displaystyle\bigwedge_{(n,m)\in unk(t,\prec)} b_{(t,\prec)} = 0 \end{array}$$

The formula χ_G is then defined as

$$\begin{array}{rcl} \chi_G & \equiv & \wedge\ c = 0 \ \wedge \displaystyle\bigvee_{n\in I} \overline{n} \\ & & \wedge\ \Box\Big[c' = 1 - c \ \wedge \displaystyle\bigvee_{(n,m)\in\delta} taken(n,m)\Big]_{\overline{b},c} \\ & & \wedge\ \Box\Diamond\langle\mathbf{true}\rangle_{\overline{b},c} \\ & & \wedge \displaystyle\bigwedge_{A\in\mathcal{A},\zeta(A)=\mathrm{WF}} \Big(\Diamond\Box \bigvee_{n\in En(A)} \overline{n}\Big) \rightarrow \Box\Diamond(action = A) \\ & & \wedge \displaystyle\bigwedge_{A\in\mathcal{A},\zeta(A)=\mathrm{SF}} \Big(\Box\Diamond \bigvee_{n\in En(A)} \overline{n}\Big) \rightarrow \Box\Diamond(action = A) \\ & & \wedge \displaystyle\bigwedge_{(t,\prec)\in\mathcal{T}} \Box\Diamond b_{(t,\prec)} = 2 \rightarrow \Box\Diamond b_{(t,\prec)} = 0 \end{array}$$

Theorem 7.4 *Let $G = (N, I, \delta, \pi, o, \zeta)$ be some predicate diagram. For any trace $\sigma = s_0\, s_1 \ldots$ through G there exists a behavior $\overline{\sigma} = \overline{s_0 s_1} \ldots$ that satisfies χ_G such that $\overline{s_i}[\![b_P]\!] = s_i[\![P]\!]$.*

Proof. Define $\overline{s_i}$ as indicated in the assertion with $\overline{s_i}[\![c]\!] = 0$ if i is even and $\overline{s_i}[\![c]\!] = 1$ otherwise. The structure of χ_G encodes the conditions of definition 7.2 for σ being a trace through G. The validity of the last conjunct is ensured because all relations $\prec \in O$ are well-founded orderings and the annotations are respected by σ. So suppose that $\overline{\sigma} \models \Box\Diamond b_{(t,\prec)} = 2$. The next-state relation of χ_G, the definition $\overline{s_i}[\![b_P]\!] = s_i[\![P]\!]$, the definition of $taken(n,m)$, and the definition of a trace imply that $s_{i+1}[\![t]\!] \prec s_i[\![t]\!]$ holds for infinitely many $i \in \mathbb{N}$. Because $\prec \in O$ is a well-founded ordering, this implies that $s_{i+1}[\![t]\!] \not\preceq s_i[\![t]\!]$ must holds for infinitely many $i \in \mathbb{N}$, and therefore, by analogous reasoning it follows that $\overline{\sigma} \models \Box\Diamond b_{(t,\prec)} = 0$. Q.E.D.

Theorem 7.4 implies that the validity of $Spec \rightarrow F$, for a TLA* formula F built from the predicates in $\mathcal{P}$, can be established by checking that $\chi_G \rightarrow \overline{F}$, for some predicate diagram G that conforms to $Spec$. For example, from the predicate diagram for the "dining mathematicians" specification shown in figure 7.1 we may verify the properties

$$\begin{array}{llll} (Pos) & \Box(n \in Nat \setminus \{0\}) & (Excl) & \Box\neg(c_0 = \text{"e"} \wedge c_1 = \text{"e"}) \\ (Live_0) & \Box\Diamond(c_0 = \text{"e"}) & (Live_1) & \Box\Diamond(c_1 = \text{"e"}) \end{array}$$

On the other hand, counter-examples produced by the model checker need not correspond to actual system runs, because not every behavior $\overline{\sigma}$ that satisfies χ_G need correspond to a "concrete" behavior satisfying the original specification $Spec$ since detail has been lost in the abstraction. Nevertheless, counter-examples that do not correspond to concrete runs can be helpful in order to refine the abstraction, for example by adding ordering annotations. Obviously, the size of diagrams that can be effectively analyzed in this way is mostly limited by the number of fairness conditions and ordering annotations present in the diagram.

7.4 Generating predicate diagrams

Theorem 7.3 can be used to justify that a given predicate diagram conforms to a given specification, but it may generate a number of proof obligations that is quadratic in the number of nodes. It is therefore desirable to construct predicate diagrams semi-automatically from a specification whenever this is possible. Conceptually, predicate diagrams act as intermediate descriptions and "interpolate" between system specifications and properties. We therefore distinguish between bottom-up constructions of diagrams that start from system specifications, and top-down constructions that start from properties. This section describes techniques that implement the bottom-up approach by abstract evaluation of specifications; section 7.5 explores top-down generation via refinement of diagrams.

7.4.1 Basic setup

Traditionally, the relationship between concrete and abstract states that underlies abstract interpretation is described by a Galois connection. Recall that a Galois connection between two partially ordered sets $(L_1, \sqsubseteq_1)$ and $(L_2, \sqsubseteq_2)$ is given by two mappings

$$(L_1, \sqsubseteq_1) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (L_2, \sqsubseteq_2)$$

such that $x_1 \sqsubseteq_1 \gamma(x_2)$ holds iff $\alpha(x_1) \sqsubseteq_2 x_2$, for all $x_1 \in L_1$ and $x_2 \in L_2$. In our setting, elements of L_1 are sets of ("concrete") states and L_2 is the Boolean algebra $\mathcal{B}(\mathcal{P})$ whose atoms are the predicates in $\mathcal{P}$. The abstraction function α returns the set of predicates true or false of a set of states, and the concretization function γ produces the set of states that are models of a set of (negated) predicates.

Given an abstract state $n \in \mathcal{B}(\mathcal{P})$, the main problem is to compute an abstract representation of the set of successor states of the states in $\gamma(n)$ with respect to some action A (including the next-state relation $Next$) of the given specification.

Definition 7.5 *For $m, n \in \mathcal{B}(\mathcal{P})$ and an action $A \in \mathcal{A}$, we say that n is an* abstract successor *of m iff for all states s, t, if $s \in \gamma(m)$ and $(s,t) \models A$, then $t \in \gamma(n)$.*

Because we may alternately interpret abstract states as elements of $\mathcal{B}(\mathcal{P})$ and as predicates over concrete states, the above definition can be restated as requiring

$$\models m \wedge A \rightarrow n'$$

Since nodes of predicate diagrams are sets of $\overline{\mathcal{P}}$, interpreted conjunctively, we may assume m to be a conjunction of literals, and try to compute some successor n in disjunctive normal form.

As the first step of the abstract evaluation of A from m, we try to evaluate those subformulas of A that contain only unprimed variables to either true or false in order to simplify the action formula. This amounts to abstractly evaluating the guards of the action. In the second step, we try to evaluate as many formulas P', for $P \in \mathcal{P}$, as possible in order to assemble information about the predicates that are true or false after the action has been executed. Throughout, we maintain A and the results of simplification in disjunctive normal form.

Let us first describe the abstract evaluation procedure for monadic predicates. If $\mathcal{P}$ contains k monadic predicates $P_1(x), \ldots, P_k(x)$ that contain the same (concrete) state variable x, we let the abstract states contain k state variables $\overline{x_1}, \ldots, \overline{x_k}$ such that $\overline{x_i}$ takes values P_i or *not*-P_i. For example, the predicate $t_1 = 0$ might be represented by the variable $\overline{t_1} \in \{zero, \textit{not-zero}\}$. Now, every unprimed occurrence of variable x in A replaced by the value assigned to $\overline{x_i}$ by the abstract source state m, for every $\overline{x_i}$ that appears in m. User-supplied rewrite rules are used to simplify the resulting expressions. The evaluation is successful if the resulting formula is of the form

$$res \;\equiv\; \bigvee_p \bigwedge_q x' = e_{pq}$$

for some expressions e_{pq} that contain the "abstract values" P_i and *not*-P_i. The set of abstract successor states is extracted from res by picking those subformulas for which e_{pq} is either P_i or *not*-P_i.

Example 7.6 As our running example for the generation of predicate diagrams we will consider the atomic, two-process version of the Bakery algorithm whose TLA specification appears in figure 6.1. Consider the evaluation of the next-state action $Next$ from the state

$$m = \{pc_1 = \text{“l0”}, pc_2 = \text{“m1”}, \overline{t_1} = zero, \overline{t_2} = zero\}$$

(Note that for these examples we slightly generalize the description in the main text in that we allow abstract values to be drawn from any finite set instead of just the Booleans. This generalization is also supported by the prototype tool that we have developed and that is described in [26].)

Suppose that we are working with a set of rewrite rules that include

$$\begin{array}{rclcrcl}
zero + 1 & \rightarrow & \textit{not-zero} & \quad & \textit{not-zero} + 1 & \rightarrow & \textit{not-zero} \\
zero = zero & \rightarrow & true & & zero = \textit{not-zero} & \rightarrow & false \\
zero \leq zero & \rightarrow & true & & zero \leq \textit{not-zero} & \rightarrow & true \\
\textit{not-zero} \leq zero & \rightarrow & false & & & &
\end{array}$$

Observe that expressions such as $\textit{not-zero} = \textit{not-zero}$ or $\textit{not-zero} \leq \textit{not-zero}$ cannot be simplified with the help of these rules.

An application of the abstract evaluation procedure outlined above produces the two successor states

$$\begin{array}{l}
\{pc_1 = \text{“l1”}, pc_2 = \text{“m1”}, \overline{t_1} = zero, \overline{t_2} = zero\} \\
\{pc_1 = \text{“l0”}, pc_2 = \text{“m2”}, \overline{t_1} = zero, \overline{t_2} = \textit{not-zero}\}
\end{array}$$

that correspond to the two possible steps that can be performed according to the protocol from state m.

Given a specification $Spec \equiv Init \wedge \Box[Next]_v \wedge L$ where we assume that v contains all flexible variables that appear in the specification, we construct a predicate diagram G as follows: first $Init$ is rewritten to disjunctive normal form, producing a set I of abstract initial states. Starting from I, we repeatedly apply abstract evaluation for $Next$ and for any action such that $\mathrm{WF}_v(A)$ or $\mathrm{SF}_v(A)$ appears as a conjunct in L in order to arrive at the set N of abstract states and the relations δ_A for the chosen actions. Assuming that all rewrite rules are correct, the resulting predicate diagram is guaranteed to satisfy the conditions of theorem 7.3.

The procedure described above can be extended to sets $\mathcal{P}$ that contain other than just monadic predicates, but it becomes somewhat more tedious to describe and implement because it can no longer simply be based on syntactic substitution. In those cases, we prefer to rely on the strengthening technique described in section 7.4.3.

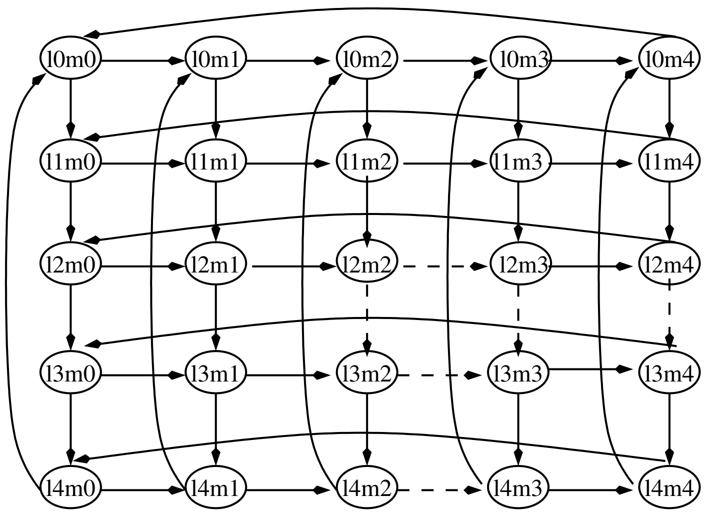


Figure 7.2: Predicate diagram for Bakery with *maybe* edges.

7.4.2 "Maybe" edges

If the abstraction is not fine enough, it is not always possible to successfully evaluate an action A with respect to a source state m, because some of the guards cannot be reduced to true or false. In such a situation, we can try to compute an approximation to the exact abstraction, introducing *maybe* edges. Any property proved from the resulting approximation is still guaranteed to hold of the original specification $Spec$. On the other hand, we may be able to eliminate *maybe* edges introducing an "on-the-fly" refinement of the abstraction with the help of an automatic prover. Alternatively, we may introduce non-determinism as allowed in condition (2) of theorem 7.3.

Suppose we choose $\{zero, not\text{-}zero\}$ as the abstract domain for the variables $\overline{t_1}$ and $\overline{t_2}$ of the Bakery algorithm. Then the abstract animation based on the evaluation rules of section 7.4.1 will fail because the expressions $not\text{-}zero \leq not\text{-}zero$ that appear in the guards of actions Try_1 and Try_2 cannot be evaluated. Our first approximation is to assign the value $maybe$ to such uninterpreted expressions:

$$not\text{-}zero \leq not\text{-}zero \quad \rightarrow \quad maybe$$

"Maybe" values are propagated using rewrite rules such as

$$\begin{aligned} maybe \wedge maybe &\rightarrow maybe \\ false \wedge maybe &\rightarrow false \\ \neg maybe &\rightarrow maybe \end{aligned}$$

Successor states can be extracted as described above even in the presence of *maybe* conjuncts, but we remember such situations and indicate edges obtained in this way using dashed edges. For the Bakery specification, we obtain the graph shown in figure 7.2 (the names of the nodes indicate the control locations of the processes; the abstract ticket values are $zero$ when control is at l_0 or l_1, resp. at m_0 or m_1, and $not\text{-}zero$ otherwise). The diagram gives a clear indication of the progress of the two processes with the first process moving in the vertical and the second in the horizontal direction.

The predicate diagram represents an approximation of the Bakery specification. It conforms to the specification, because every behavior of the Bakery algorithm is a trace through the diagram. However, it is too weak: it cannot be used to prove the invariant

$$Bakery \rightarrow \Box\neg(pc_1 = \text{“l3”} \ \wedge \ pc_2 = \text{“m3”})$$

that asserts mutual exclusion for the two processes. However, every path leading to the abstract state l3m3 contains a dashed edge, and these edges indicate opportunities for refining the approximation by reconsidering the transitions in view of the concrete specification. We now describe a technique that can perform such a refinement during the computation of the diagram.

7.4.3 “On-the-fly” strengthening

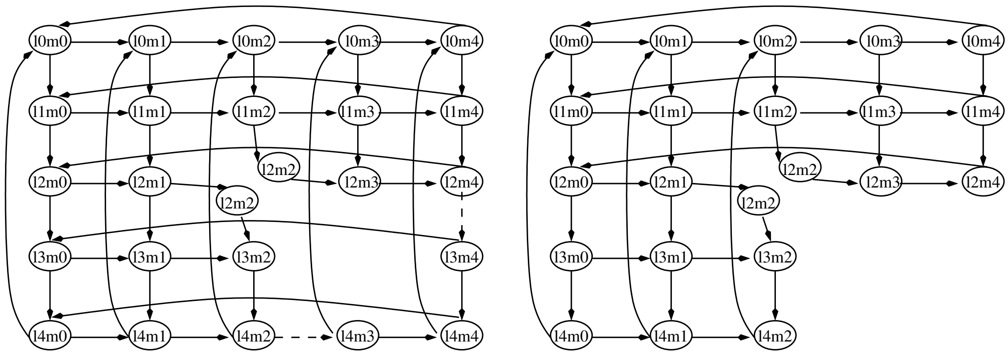
Maybe edges are introduced when predicates that appear in the concrete-level next-state relation (such as $t_1 \leq t_2$ in the Bakery example) cannot be interpreted in the abstract states. We suggest to reconsider the offending transitions based on the predecessors of the abstract source state. More precisely, suppose that we cannot interpret a sub-formula Q in an abstract state m during the evaluation of action A. We may then try to evaluate Q' with respect to the formula $n \wedge A \wedge m'$, for every predecessor n of m. In other words, we try to prove either of

$$n \wedge A \wedge m' \ \rightarrow \ Q' \qquad \text{or} \qquad n \wedge A \wedge m' \ \rightarrow \ \neg Q'$$

The construction of successor nodes ensures that $n \wedge A \wedge m'$ must be consistent, and therefore at most one of the implications can be provable. We consider the three following cases:

	(1)	(2)	(3)
$n \wedge A \wedge m' \rightarrow Q'$	proved	unproved	unproved
$n \wedge A \wedge m' \rightarrow \neg Q'$	unproved	proved	unproved

In the first and second case, we may add Q resp. $\neg Q$ to the label of m (if m has more than one predecessor, the node is split into two copies). If all subformulas



(a) On-the-fly strengthening. (b) Forward propagation.

Figure 7.3: Predicate diagrams for Bakery after strengthening.

that originally evaluated to $maybe$ can be decided one way or another, the *maybe* edge can be replaced by a standard transition.

In the case of the Bakery example we can prove

$$\begin{aligned}&\mathrm{l2m1} \wedge Next \wedge \mathrm{l2m2}' \rightarrow \neg(y_1' \leq y_2')\\&\mathrm{l1m2} \wedge Next \wedge \mathrm{l2m2}' \rightarrow y_1' \leq y_2'\end{aligned}$$

Consequently, state l2m2 is split into two states and ingoing transitions are adjusted accordingly. Continuing the abstract evaluation, both *maybe* edges leading to node l3m3 disappear, and mutual exclusion can be proven. In effect, we have "discovered" an additional predicate that should have been included in the set $\mathcal{P}$ of predicates to abstract on.

In the implementation of the abstract evaluation procedure described in [26], we remember the subformulas of the concrete-level action that evaluated to $maybe$. If such expressions remain after rewriting completes, the tool calls the automatic prover Simplify [35] in order to try and solve the resulting proof obligations, and splits the node as necessary. Applying this modified generation procedure to the Bakery specification, we arrive at the predicate diagram schematically shown in figure 7.3(a). In this diagram, the node at which both $pc_1 =$ "l2" and $pc_2 =$ "m2" are true has been split into two nodes that differ in the value assigned to the predicate $y_1 \leq y_2$, and the node l3m3 has been eliminated. In particular, mutual exclusion can now be established, although the diagram still contains two *maybe* edges.

Because of these remaining *maybe* edges, we can still not prove liveness properties about the Bakery algorithm. However, the edges can be eliminated by forward propagation of the inferred properties in a similar manner. In this way we obtain the predicate diagram shown in figure 7.3(b) for the Bakery example which precisely represents the behaviors of the specification modulo the chosen abstraction. Running the model checker on this last diagram confirms the liveness property

$$Bakery \rightarrow (pc_1 = \text{“l1”} \leadsto pc_1 = \text{“l3”})$$

and symmetrically for the other process.

Similar graphs have been produced manually, for example, in [55], but our evaluation procedure has been able to generate this predicate diagram with the user only indicating that zero and non-zero values for the “ticket” variables should be distinguished.

7.5 Refinement of predicate diagrams

Having seen applications of predicate diagrams to support “bottom-up” system verification, we now study refinement notions on diagrams that can support the development of systems: starting from a first, coarse abstraction that represents overall system behavior, more and more detail is added until the system specification can be directly “read off” the final diagram. As in the case of verification, the semantic refinement notion is given by trace inclusion; in particular, closure of traces under finite stuttering allows to add more detail on lower levels of abstraction. As in the case of verification, we are therefore mostly facing a methodological problem: we need definitions that are both flexible in that they support non-trivial refinements and tractable in that proof obligations are either local or amenable to model checking. Our proposed solution is to restrict ourselves to structural refinements of diagrams as far as safety properties are concerned, but allow liveness conditions to be refined very liberally. Note that we deliberately sacrifice the completeness of our refinement notion w.r.t. trace inclusion in favor of a clear structure that supports documentation of the development process.

7.5.1 Behavioral refinement

To simplify matters, we first assume that the refining, “concrete” diagram G^1 interprets all predicates and actions that occur in the refined, “abstract” diagram G^2. In other words, the sets of predicates, action names, and ordering annotations that appear in G^1 are supersets of the corresponding sets in G^1. A more general definition that allows predicates of G^2 to be represented differently in G^1, and therefore allows for change of representation of the underlying data, will be considered in section 7.5.3.

Definition 7.7 *Assume given two predicate diagrams $G^1 = (N^1, I^1, \delta^1, o^1, \zeta^1)$ over $\mathcal{P}^1$ and $\mathcal{A}^1$ and $G^2 = (N^2, I^2, \delta^2, o^2, \zeta^2)$ over $\mathcal{P}^2$ and $\mathcal{A}^2$ where $\mathcal{P}^1 \supseteq \mathcal{P}^2$ and $\mathcal{A}^1 \supseteq \mathcal{A}^2$, and let $f : N^1 \to N^2$. We say that G^1* structurally refines *G^2* w.r.t. *f iff all the following conditions hold:*

1. $f(I^1) \subseteq I^2$
2. $\models n \to f(n)$ *holds for every node $n \in N^1$.*
3. *For all $n, m \in N^1$ and all $A \in \mathcal{A}^1$ such that $(n,m) \in \delta^1_A$:*
   (a) *if $A \in \mathcal{A}^2$ then $(f(n), f(m)) \in \delta^2_A$, and*
   (b) *if $A \in \mathcal{A}^1 \setminus \mathcal{A}^2$ then $(f(n), f(m)) \in \delta^2_=$.*
4. *For all $n, m \in N^1$, all $A \in \mathcal{A}^1$ such that $(n,m) \in \delta^1_A(n,m)$, all terms t and relations $\prec \in O^=$:*
   (a) *$(t, \prec) \in o^1(n,m)$ whenever $(t, \prec) \in o^2(f(n), f(m))$,*
   (b) *$(t, \preceq) \in o^1(n,m)$ whenever $f(n) = f(m)$ and $(t, \prec) \in o^2(f(n), m')$ for some $m' \in N^2$.*
5. *For every run $\rho^1 = (s_0, n_0, A_0)(s_1, n_1, A_1)\ldots$ of G^1 and every action $A \in \mathcal{A}^2$ such that $\zeta^2(A) = \mathrm{WF}$, either $A_i = A$ or $f(n_i) \notin En^2(A)$ holds for infinitely many $i \in \mathbb{N}$.*
6. *For every run $\rho^1 = (s_0, n_0, A_0)(s_1, n_1, A_1)\ldots$ of G^1 and every action $A \in \mathcal{A}^2$ such that $\zeta^2(A) = \mathrm{SF}$, either $A_i = A$ holds for infinitely many $i \in \mathbb{N}$ or $f(n_i) \in En^2(A)$ for only finitely many $i \in \mathbb{N}$.*

We say that G^1 structurally refines *G^2 iff G^1 structurally refines G^2 w.r.t. some function $f : N^1 \to N^2$.*

The definition of structural refinement is based on a decomposition of nodes, formally represented by a refinement function f that for every node of the concrete diagram indicates of which abstract node it is a subnode. The obligations are stated such that conditions (1)–(4) are either purely structural or can be verified using non-temporal reasoning. On the other hand, conditions (5) and (6) are based on the runs of the diagrams and can therefore be established by model checking, using the encoding described in section 7.3.2. Note in particular that we do not require high-level fairness conditions to be syntactically preserved in the refining diagram; as we will see in the examples below, this makes our definition flexible. On the other hand, we require that transitions in G^1 preserve any ordering annotations asserted for the corresponding transitions of G^2, and that transitions of G^1

that correspond to stuttering steps in G^2 do not increase terms for which ordering annotations are present in G^1. This requirement of strict preservation could be relaxed by allowing a transition of the abstract diagram with an associated ordering constraint $(t, \prec)$ to be simulated by a sequence of transitions in the refining diagram such that each transition ensures $t' \preceq t$ and at least one transition ensures $t' \prec t$. We have chosen the stricter definition because we view ordering annotations as strong commitments that any later refinement is bound to implement.

The following theorem asserts that structural refinement ensures trace inclusion, and is therefore sound. In particular, all properties shown of G^2 remain valid for G^1. Definition 7.7 also implies that structural refinement is a preorder (i.e., reflexive and transitive).

Theorem 7.8 *If G^1 structurally refines G^2 then $tr(G^1) \subseteq tr(G^2)$.*

Proof. Assume that G^1, G^2 and f are as in definition 7.7, and that $\sigma = s_0 s_1 \ldots$ is a trace of G^1. We must prove that $\sigma \in tr(G^2)$.

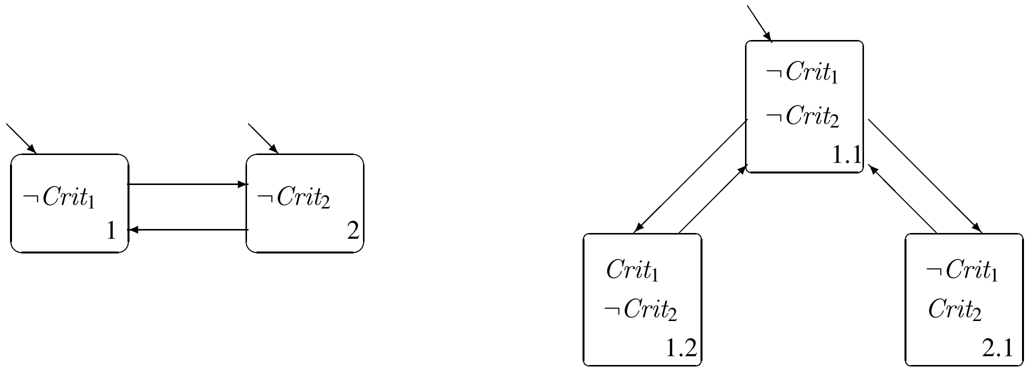
By the definition of a trace, choose some run $\rho = (s_0, n_0, A_0)(s_1, n_1, A_1)\ldots$ of G^1 based on σ. We will define a sequence $\pi = (s_0, f(n_0), B_0)(s_1, f(n_1), B_1)\ldots$ and prove that π is a run of G^2. Define the sets $\mathcal{B}_i$ by

$$\mathcal{B}_i = \{A \in \mathcal{A}^2 : (f(n_i), f(n_{i+1})) \in \delta^2_A\}$$

and let $\mathcal{B}^*_i = \mathcal{B}_i$ if $f(n_i) \neq f(n_{i+1})$, and $\mathcal{B}^*_i = \mathcal{B}_i \cup \{\tau\}$ otherwise. Condition (3) of definition 7.7 ensures that $\mathcal{B}^*_i \neq \emptyset$ holds for all $i \in \mathbb{N}$. Now choose a sequence $B_0, B_1, \ldots$ of actions $B_i \in \mathcal{B}^*_i$ such that every action $A \in \mathcal{A}^2$ that appears in infinitely many sets $\mathcal{B}^*_i$ is chosen infinitely often. (Such a choice is possible because the set $\mathcal{A}^2$ is finite.)

Since ρ is a run of G^1, we know that $n_0 \in I^1$, hence condition (1) ensures that $f(n_0) \in I^2$. Moreover, $s_i \models n_i$ for all $i \in \mathbb{N}$, and condition (2) implies $s_i \models f(n_i)$. Similarly, the choice of B_i and condition (3) ensures that either $B_i = \tau$ and $f(n_i) = f(n_{i+1})$ or $B_i \in \mathcal{A}^2$ and $(f(n_i), f(n_{i+1})) \in \delta^2_{B_i}$. Finally, using condition (4) and the fact that ρ is a run of G^1, it follows that $s_{i+1}[\![t]\!] \prec s_i[\![t]\!]$ holds whenever $(t, \prec) \in o^2(f(n_i), f(n_{i+1}))$ if $B_i \in \mathcal{A}^2$, and that $s_{i+1}[\![t]\!] \preceq s_i[\![t]\!]$ if $B_i = \tau$ and node $f(n_i)$ has an outgoing edge labelled by $(t, \prec)$.

It remains to show that π satisfies the fairness conditions associated with G^2. Thus, let $A \in \mathcal{A}^2$ be some action such that $\zeta^2(A) = \text{WF}$, and assume that $f(n_i) \in En^2(A)$ holds for all but finitely many $i \in \mathbb{N}$ (otherwise, the fairness condition is trivially satisfied). Condition (5) implies that $A_i = A$ holds for infinitely many $i \in \mathbb{N}$, and condition (3a) ensures that $A \in \mathcal{B}^*_i$ holds for infinitely many $i \in \mathbb{N}$. The choice of actions B_i implies $B_i = A$ for infinitely many $i \in \mathbb{N}$, which completes the proof. The argument for actions $A \in \mathcal{A}^2$ such that $\zeta^2(A) = \text{SF}$ is analogous, replacing "all but finitely many" by "infinitely many" and using condition (6) instead of (5). Q.E.D.



(a) Initial diagram. (b) First refinement.

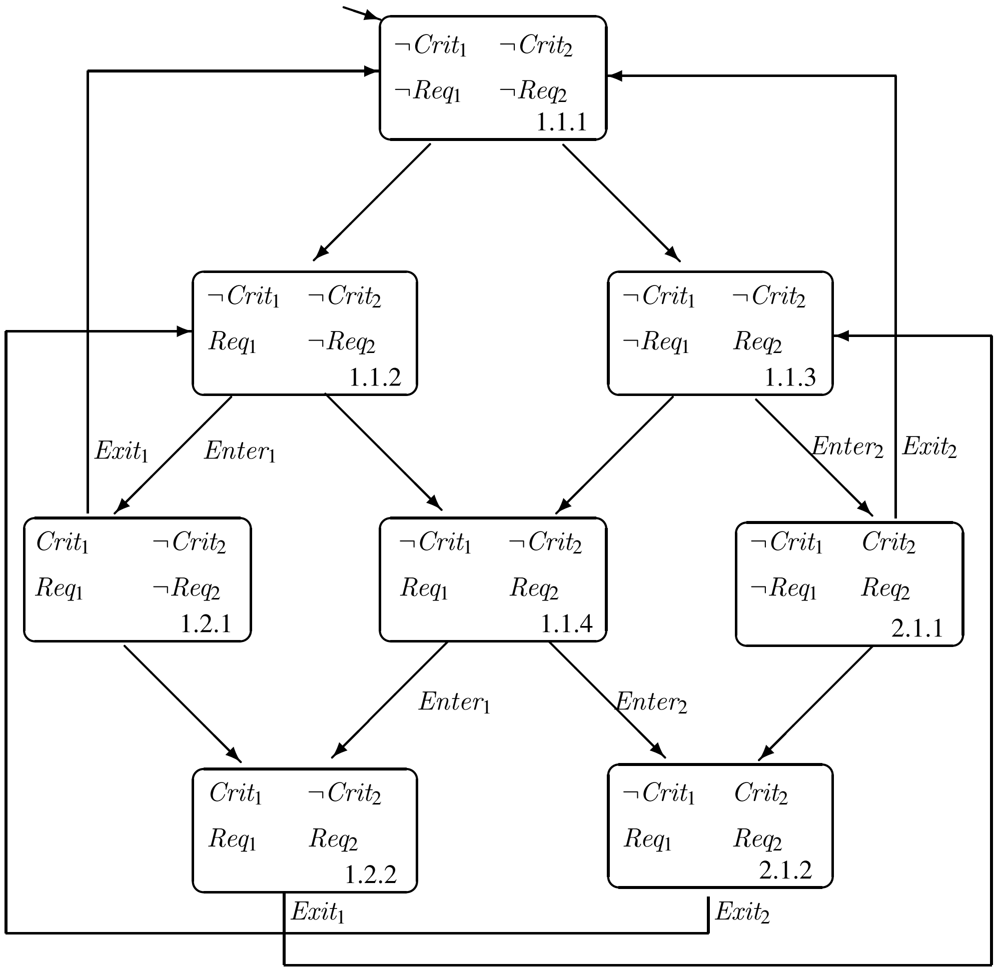
Figure 7.4: Mutual exclusion: initial diagram and first refinement.

7.5.2 Case study: mutual exclusion

We will now argue that the notion of structural refinement is flexible by performing a sequence of refinements that lead to different two-process mutual exclusion protocols, a standard, albeit simple benchmark problem in the design of reactive systems. The basic requirement is to ensure that the two processes can never simultaneously access their critical sections. The simplest predicate diagram that expresses this requirement is shown in figure 7.4(a). It consists of two nodes that represent, respectively, system states where either the first or the second process is outside of its critical section. No indication is given how mutual exclusion is achieved, nor does the predicate express any liveness properties.

A first design decision is represented by the diagram shown in figure 7.4(b), which rules out direct hand-overs of the resource from one process to the other but requires the processes to go through an intermediate state where no process is in its critical section; this is also the initial state. Here and in the following diagrams, we use a numbering convention to indicate the refinement function: a node $x.y$ of the refining diagram is mapped to node x of the abstract one. (We have arbitrarily decided to map the initial state to the abstract state 1.) It is easy to see that the second diagram is a structural refinement of the first one because every node label implies the label of the corresponding abstract node, and all concrete transitions can be mapped to either an explicit abstract transition or to a loop at the abstract level. The conditions (4)–(6) of definition 7.7 are all vacuously satisfied.

Continuing our development, we now add fairness requirements for processes that have requested the resource. We therefore add predicates Req_1 and Req_2 that indicate which processes have issued a request and split the nodes accordingly, as shown in figure 7.5. We have also introduced actions $Enter_1$, $Enter_2$, $Exit_1$, and



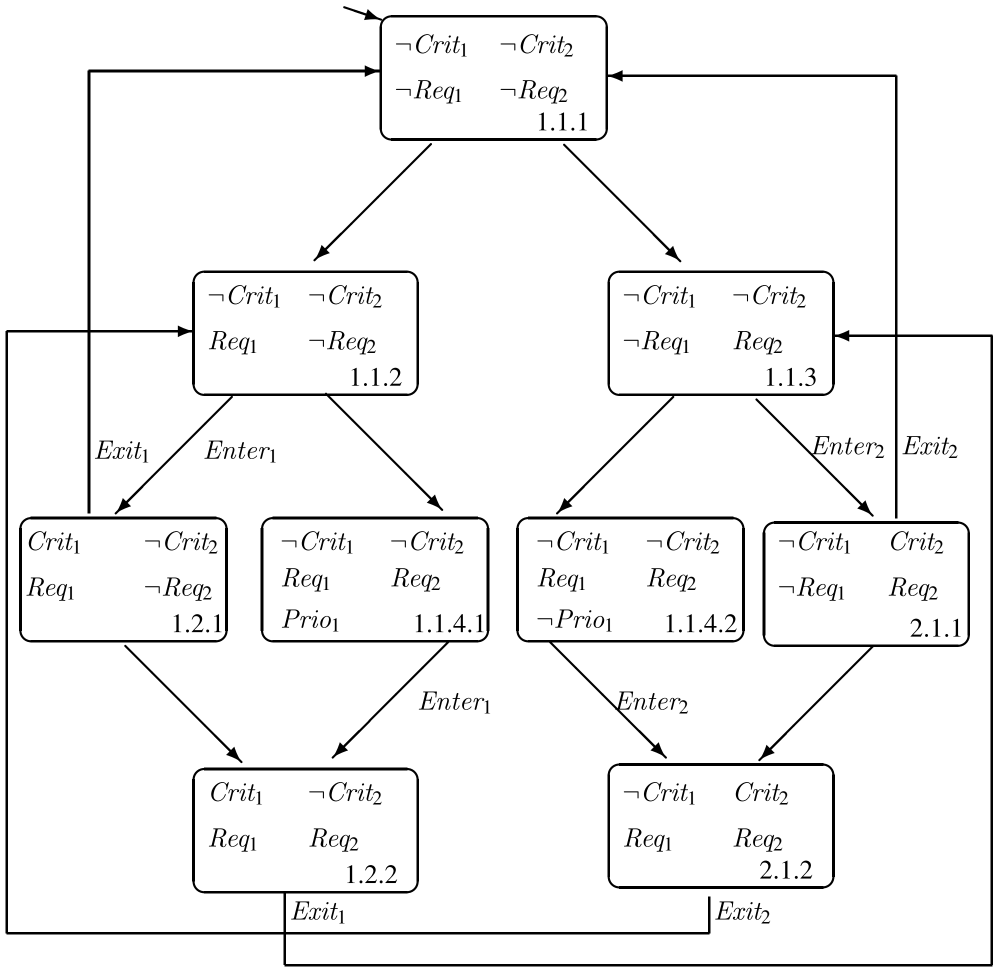
$\zeta(Enter_1) = \zeta(Enter_2) = \text{SF} \qquad \zeta(Exit_1) = \zeta(Exit_2) = \text{WF}$

Figure 7.5: Mutual exclusion: adding requests and fairness.

$Exit_2$ that model processes entering and exiting their critical sections.

It is again easy to check that the diagram of figure 7.5 structurally refines that of figure 7.4(b). In particular, transitions such as the one from node 1.1.1 to 1.1.2 where process 1 requests the resource are covered by stuttering transitions in the previous diagram. Definition 7.7 also allows to name previously unnamed transitions such as that from node 1.1.3 to node 2.1.1.

As shown in the figure, we require strong fairness of the $Enter_i$ actions, and weak fairness of the $Exit_i$ actions to ensure that conflicts are resolved in a fair way, and that no process monopolizes the resource. (Note that strengthening fairness conditions is a particular case of structural refinement.) Model checking



$$\zeta(Enter_1) = \zeta(Enter_2) = \text{WF} \qquad \zeta(Exit_1) = \zeta(Exit_2) = \text{WF}$$

Figure 7.6: Mutual exclusion: implementing fairness via priorities.

establishes that the diagram satisfies the liveness properties

$$Req_1 \leadsto Crit_1 \qquad \text{and} \qquad Req_2 \leadsto Crit_2 \tag{7.1}$$

The diagram of figure 7.5 can be seen as representing a semaphore implementation of mutual exclusion: the semaphore is taken in those states where some process is inside its critical section, and free otherwise. Interestingly, we can continue our development to obtain an implementation along the lines of Peterson's algorithm if we assign priorities to the processes and ensure that priorities change whenever both processes compete for access to the resource. We only need to split node 1.1.4 into two subnodes as shown in figure 7.6; the nodes 1.1.4.1 and 1.1.4.2 assign priority to different processes. Because there is no longer a conflict between the actions $Enter_1$ and $Enter_2$, the strong fairness requirement for these actions can be replaced by weak fairness.

It is easy to establish conditions (1)–(4) in order to prove the correctness of the refinement because the diagram of figure 7.6 allows fewer transitions than the one shown in figure 7.5. Proving that the original fairness conditions are also refined is more subtle because the nodes 1.1.4.1 and 1.1.4.2 are both mapped to the abstract node 1.1.4 where both $Enter_1$ and $Enter_2$ are enabled. However, the alternation of priority ensures that conflicts are fairly resolved, and in fact a run of the model checker confirms that condition (6) of definition 7.7 is satisfied. The predicate diagram shown in figure 7.6 therefore inherits all the desired safety and liveness properties that had been obtained for the previous diagrams.

7.5.3 Data refinement

Structural refinement as introduced in definition 7.7 ensures that every trace through the refining diagram is also a trace through the original diagram. We now present a more general refinement notion and allow the refining diagram G^1 to be based on a different set of predicates than the original diagram G^2. Consequently, there may be variables x^2 that occur in G^2 but not in G^1; in other words, the representation of data may have changed. We require the state spaces of the two diagrams to be connected by a "gluing invariant" that, intuitively, allows the abstract variables and predicates to be computed from the concrete ones. The definition ensures that for every trace through the refining diagram there exists a trace through the original diagram such that corresponding states in the traces satisfy the gluing invariant. It follows that properties established for the original diagram are preserved up to the gluing invariant. Similar notions of data refinement are familiar from refinement calculus [14] and its descendants such as Z [93] and B [9].

Definition 7.9 *Assume given two predicate diagrams $G^1 = (N^1, I^1, \delta^1, o^1, \zeta^1)$ over $\mathcal{P}^1$ and $\mathcal{A}^1$ and $G^2 = (N^2, I^2, \delta^2, o^2, \zeta^2)$ over $\mathcal{P}^2$ and $\mathcal{A}^2$ where $\mathcal{A}^1 \supseteq \mathcal{A}^2$, as well as a state predicate I. Let x^2 be a tuple of all variables that occur in $\mathcal{P}^2$, but not in $\mathcal{P}^1$, and let $f : N^1 \to N^2$. We say that G^1* structurally refines *G^2* up to *I* w.r.t. *f *iff all of the following conditions hold:*

1. $f(I^1) \subseteq I^2$,
2. $\models n \to \exists x^2 : I \wedge f(n)$ *holds for every node* $n \in N^1$.
3. *For all* $n, m \in N^1$ *and all* $A \in \mathcal{A}^1$ *such that* $(n,m) \in \delta^1_A$*:*
   (a) *if* $A \in \mathcal{A}^2$ *then* $(f(n), f(m)) \in \delta^2_A$*, and*
   (b) *if* $A \in \mathcal{A}^1 \setminus \mathcal{A}^2$ *then* $(f(n), f(m)) \in \delta^2_=$.

4. *For all* $n, m \in N^1$ *and all* $A \in \mathcal{A}^1$ *such that* $(n,m) \in \delta^1_A$*:*

   (a) *for all* $(t_2, \prec_2) \in o^2(f(n), f(m))$ *there exists some* $(t_1, \prec_1) \in o^1(n,m)$ *such that*

   - $\models n \wedge m' \wedge I \wedge I' \wedge t_1' \prec_1 t_1 \rightarrow t_2' \prec_2 t_2$ *and*
   - $\models n \wedge n' \wedge I \wedge I' \wedge t_1' \preceq_1 t_1 \rightarrow t_2' \preceq_2 t_2$,

   (b) *if* $f(n) = f(m)$ *and* $(t_2, \prec_2) \in o^2(f(n), m')$ *holds for some* $m' \in N^2$ *then there exists some* $(t_1, \prec_1) \in o^1(n,m)$ *such that*

   $$\models n \wedge m' \wedge I \wedge I' \wedge t_1' \prec_1 t_1 \rightarrow t_2' \preceq_2 t_2$$

5. *For every run* $\rho^1 = (s_0, n_0, A_0)(s_1, n_1, A_1)\ldots$ *of* G^1 *and every action* $A \in \mathcal{A}^2$ *such that* $\zeta^2(A) = \mathrm{WF}$*, either* $A_i = A$ *or* $f(n_i) \notin En^2(A)$ *holds for infinitely many* $i \in \mathbb{N}$.

6. *For every run* $\rho^1 = (s_0, n_0, A_0)(s_1, n_1, A_1)\ldots$ *of* G^1 *and every action* $A \in \mathcal{A}^2$ *such that* $\zeta^2(A) = \mathrm{SF}$*, either* $A_i = A$ *holds for infinitely many* $i \in \mathbb{N}$ *or* $f(n_i) \in En^2(A)$ *for only finitely many* $i \in \mathbb{N}$.

We say that G^1 structurally refines G^2 up to I *iff* G^1 *structurally refines* G^2 *up to* I *w.r.t. some function* $f : N^1 \rightarrow N^2$.

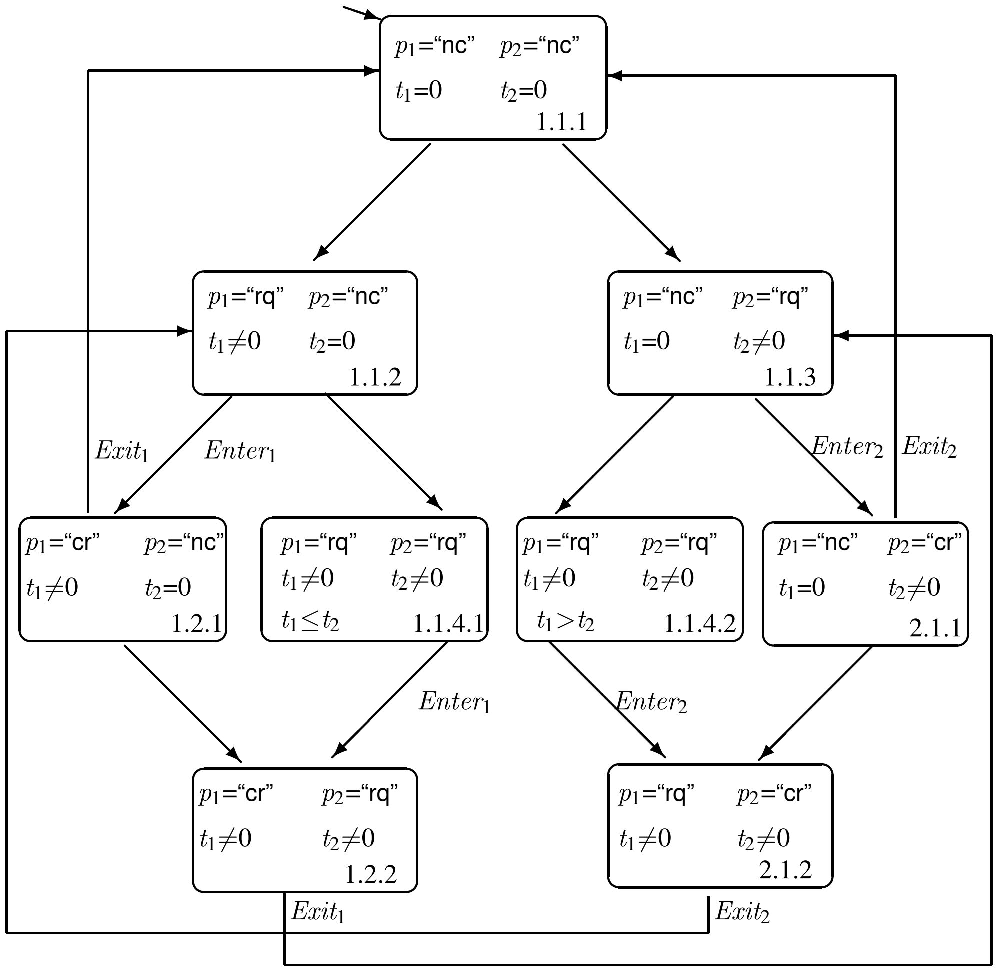
We still require the actions of the original diagram to appear in the refining diagram. This requirement could be dropped, in particular for actions without fairness conditions, but this would complicate the statement of definition 7.9, and does not appear very useful—recall that only action names appear in predicate diagrams, and that an implementation may assign the same action formula to different action names. More ambitious generalizations would allow the abstract variables x^2 to be related to concrete-level variables by auxiliary (e.g., history and prophecy) variables as discussed in section 4.7 instead of a simple gluing invariant L. Our current experience is too limited to determine a useful and tractable formulation of such a more flexible definition.

The correctness property associated with structural refinement up to a gluing invariant is stated in the following theorem.

Theorem 7.10 *Assume that* G^1 *structurally refines* G^2 *up to* I*, and let* x^2 *again denote a tuple of all variables that occur in* G^2*, but not in* G^1*.*

1. *For every trace* $\sigma = s_0 s_1 \ldots$ *through* G^1 *there exists a trace* $\tau = t_0 t_1 \ldots$ *through* G^2 *such that* $\tau =_{x^2} \sigma$ *and such that* $t_i[\![I]\!]$ *holds for all* $i \in \mathbb{N}$.

2. *If* G^2 *satisfies formula* F *then* G^1 *satisfies* $\exists\, x^2 : \Box I \wedge F$.



$$\zeta(Enter_1) = \zeta(Enter_2) = \mathrm{WF} \qquad \zeta(Exit_1) = \zeta(Exit_2) = \mathrm{WF}$$

Figure 7.7: Mutual exclusion: two-process Bakery algorithm (atomic version).

Proof.

1. The proof follows the argument used in the proof of theorem 7.8, except that for the definition of the corresponding run of G^2 some x^2-variant t_i of every state s_i needs to be chosen such that $t_i \models L$. The existence of such states is ensured by condition (2). The actions B_i are chosen as before, and the proof obligations for the ordering annotations in G^1 ensure that the run of G^2 meets all required conditions.

2. An immediate consequence of assertion 1. Q.E.D.

To illustrate the application of definition 7.9, we continue the development of mutual-exclusion protocols carried out in section 7.5.2 to derive an atomic version of Lamport's two-process Bakery algorithm. Figure 7.7 shows a diagram with the same structure as the diagram of figure 7.6, but with a different labelling of the nodes and some additional action names. (Think of p_1 and p_2 as program counters whose values correspond to “non-critical”, “requesting”, and “critical”.) As before, we require weak fairness of the $Enter_i$ and $Exit_i$ actions.

It is straightforward to prove that the diagram of figure 7.7 structurally refines the diagram of figure 7.6 up to the gluing invariant I defined as

$$\begin{aligned}
&\wedge\ (Crit_1 \equiv p_1 = \text{“cr”}) \wedge (Crit_2 \equiv p_2 = \text{“cr”})\\
&\wedge\ (Req_1 \equiv t_1 \neq 0) \wedge (Req_2 \equiv t_2 \neq 0)\\
&\wedge\ Prio_1 \equiv (t_1 \leq t_2)
\end{aligned}$$

In particular, theorem 7.10.2 and simple temporal reasoning implies that the properties

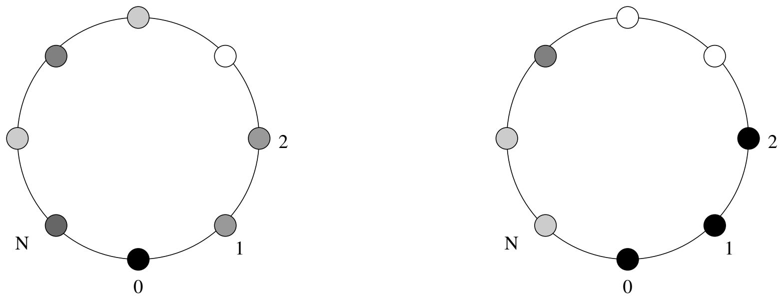
$$t_1 \neq 0 \leadsto p_1 = \text{“cr”} \qquad \text{and} \qquad t_2 \neq 0 \leadsto p_2 = \text{“cr”} \tag{7.2}$$

can be inferred for the diagram of Fig. 7.7 from the liveness properties (7.1) verified for the semaphore implementation of mutual exclusion. In other words, whenever a process has “drawn a ticket”, it is guaranteed to eventually enter its critical section.

To arrive at an actual implementation, we must find an assignment of action formulas to action names that satisfies the conditions of theorem 7.3. One possible solution is to let the Req_i actions assign the ticket value of the other process, incremented by one, to the ticket of the own process, and to let the $Exit_i$ actions reset the ticket value to 0. This completes our derivation of the atomic version of Lamport’s Bakery algorithm.

7.5.4 Case study: self-stabilizing protocol

In a seminal paper [36], Dijkstra introduced the concept of self-stabilizing systems and presented three algorithms that achieve self-stabilization in a ring of processors. As a somewhat more substantial case study, we will apply the format of predicate diagrams to verify one of Dijkstra’s algorithms. Specifically, starting from an abstract diagram G_{abs} that expresses the concept of self-stabilization we will derive a refinement G_{conc} that conforms to a specification of Dijkstra’s protocol. In this example, the correctness of the refinement is justified by model checking, and theorem 7.3 is applied to establish conformance via non-temporal theorem proving. In contrast to a previous “flat” verification [86] of the protocol in PVS, our proof has a clearer structure and also requires less user interaction because part of the verification effort is delegated to model checking. The diagrammatic presentation of the abstraction helps to clarify the structure of the



(a) Initial configuration. (b) Later configuration.

Figure 7.8: Evolution of Dijkstra's protocol.

proof and provides a natural interface between the model checker and the interactive theorem prover. The refinement conditions stated in definition 7.7 are flexible enough to justify the implementation of fairness conditions at the higher level by a combination of lower-level fairness conditions and well-founded orderings.

A system is self-stabilizing if it will reach some "stable" state (and then remain "stable"), no matter what state it starts in. Self-stabilizing protocols are useful for initialization and for error-recovery after faults. Dijkstra [36] introduced the problem at the hand of a ring of $N+1$ processors numbered 0 to N. Each processor i is equipped with a register $v[i]$ that can hold values in the range $0, \ldots, M$ where $M \geq N$. Figure 7.8(a) illustrates a possible configuration where different shades of grey represent different register values.

The processors operate according to the following rules: processor 0 can make a move whenever its register holds the same value as that of its left-hand neighbor, processor N. It then increments its register (modulo $M+1$). Any other processor $i+1$ can make a move when the value in its register is different from the value held by its left-hand neighbor, processor i. Processor $i+1$ then copies the value of $v[i]$ into its own register. The ability to make a move is interpreted as the possession of a token; the system is in a stable state when exactly one processor has a token. A TLA specification (written in a suitable first-order language) of the protocol appears in figure 7.9.

Fig. 7.8(b) shows a configuration that can be reached from the configuration of Fig. 7.8(a) after a few steps. Observe that at least one processor is enabled (i.e., has a token) in any configuration: for if all non-zero processors are disabled then $v[i] = v[i+1]$ holds for all $0 \leq i < N$. Therefore, $v[0] = v[N]$, implying that processor 0 is enabled. Stable configurations are those in which exactly one processor can move, which is true precisely if for some i, all processors $j \leq i$ hold the value $v[0]$ while all processors $j > i$ hold the value $v[N]$. In particular, configurations where all registers hold the same value are stable. It is also easy to

$$
\begin{array}{rcl}
\mathit{SITES} & \triangleq & \{\, x \in \mathbb{N} : 0 \leq x \wedge x \leq N \,\} \\
\mathit{NUM} & \triangleq & \{\, x \in \mathbb{N} : 0 \leq x \wedge x \leq M \,\} \\
\mathit{Zero} & \triangleq & \wedge\ v[0] = v[N] \\
 & & \wedge\ v' \in [\mathit{SITES} \rightarrow \mathit{NUM}] \\
 & & \wedge\ v'[0] = v[0] + 1 \text{ mod } (M+1) \\
 & & \wedge\ \forall j \in \mathit{SITES} : j \neq 0 \rightarrow v'[j] = v[j] \\
\mathit{NonZero}(i) & \triangleq & \wedge\ v[i+1] \neq v[i] \\
 & & \wedge\ v' \in [\mathit{SITES} \rightarrow \mathit{NUM}] \\
 & & \wedge\ v'[i+1] = v[i] \\
 & & \wedge\ \forall j \in \mathit{SITES} : j \neq i+1 \rightarrow v'[j] = v[j] \\
\mathit{Init} & \triangleq & v \in [\mathit{SITES} \rightarrow \mathit{NUM}] \\
\mathit{Next} & \triangleq & \mathit{Zero} \vee \exists i \in \mathit{SITES} : i \neq N \wedge \mathit{NonZero}(i) \\
\mathit{Spec} & \triangleq & \mathit{Init} \wedge \Box[\mathit{Next}]_v \wedge \mathrm{WF}_v(\mathit{Next})
\end{array}
$$

Figure 7.9: A TLA specification of Dijkstra's protocol.

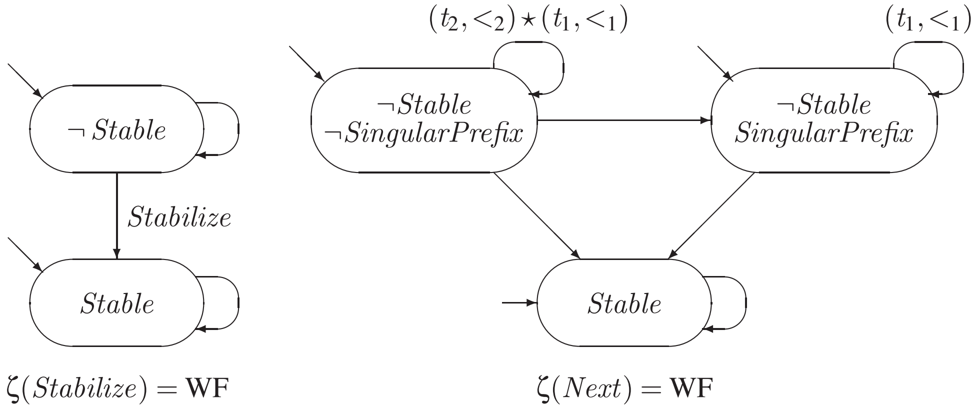
see that whenever a stable state has been reached, the token will circulate counterclockwise through the ring, and that all subsequent configurations will also be stable.

It was only a decade after the publication of the protocol [37] that Dijkstra published a proof demonstrating its correctness, acknowledging that it was not entirely trivial. In 1998, Qadeer and Shankar [86] published a machine-checked verification of the algorithm using the interactive theorem prover PVS. They reported considerable difficulty both in constructing a convincing informal proof and in its formalization. We now study another proof that relies on refinement of predicate diagrams. We will see that part of the verification effort can be automated through the use of model checking.

The property of self-stabilization can be abstractly represented by the predicate diagram G_{abs} of figure 7.10(a): initially, the system can either be in a stable or unstable state. Transitions from a stable to an unstable state are not allowed; eventual stabilization is enforced by placing a fairness condition on an action named $\mathit{Stabilize}$ that causes a transition from unstable to stable states. A run of the model checker on the encoding of the diagram confirms:

Proposition 7.11 *Every trace through G_{abs} satisfies $\Diamond\Box \mathit{Stable}$.*

Diagram G_{abs} concisely represents the desired property, but it is too simple. Formally, it does not conform to the formula Spec of figure 7.9 that represents Dijkstra's protocol: there is no action corresponding to $\mathit{Stabilize}$ that would be



(a) Abstract view G_{abs}.

(b) Refinement G_{conc} for Dijkstra's protocol.

Figure 7.10: Predicate diagrams to express self-stabilization.

enabled in all unstable states and lead to immediate stabilization. We have to find a refinement of G_{abs} that conforms to $Spec$.

Informally, the correctness of Dijkstra's protocol can be explained as follows. Consider first a configuration where processor 0 holds a value that does not occur in any other register (such a configuration is depicted in figure 7.8(a)) or, more generally, a configuration where $v[0] = v[1] = \ldots = v[i]$ holds for some $i \leq N$, but $v[j] \neq v[0]$ holds for all j where $i < j \leq N$. We say that such a configuration has a *singular prefix*. In such a configuration, processor 0 cannot move unless $v[0] = v[N]$, which means that the prefix extends around the entire ring and the system has stabilized. The non-zero processors in the prefix are disabled in such a configuration. On the other hand, moves of processors outside the prefix do not introduce fresh values and will eventually extend the singular prefix around the ring.

It is therefore enough to show that some configuration with a singular prefix will be reached from any initial configuration, unless a stable configuration is reached before. Now, observe that the number $M+1$ of possible register values has been assumed to be at least the number of processors. By the pigeonhole principle it follows that some value $r \in \{0, \ldots, M\}$ does not initially occur among $v[1], \ldots, v[N]$. Assuming that processor 0 moves infinitely often, thereby incrementing its register, there will be a first subsequent configuration where $v[0] = r$. In that configuration it must still be the case that $v[i+1] \neq r$ for all i because only moves of processor 0 introduce fresh values; thus we have reached a configuration with a singular prefix. It thus remains to show that processor 0 is guaranteed to move infinitely often. But assume that processor 0 does not move from some state on: then the actions of the non-zero processors will ensure that $v[0]$ spreads

$$
\begin{array}{rcl}
\mathit{stable}(i) & \triangleq & \forall j \in \mathit{SITES} : \land\, j \leq i \rightarrow v[j] = v[0] \\
 & & \qquad\qquad\qquad\;\; \land\, j > i \rightarrow v[j] = v[N] \\
\mathit{Stable} & \triangleq & \exists i \in \mathit{SITES} : \mathit{stable}(i) \\
\mathit{SP}(i) & \triangleq & \forall j \in \mathit{SITES} : \land\, j \leq i \rightarrow v[j] = v[0] \\
 & & \qquad\qquad\qquad\;\; \land\, j > i \rightarrow v[j] \neq v[0] \\
\mathit{SingularPrefix} & \triangleq & \exists i \in \mathit{SITES} : \mathit{SP}(i) \\
t_1 & \triangleq & [\, i \in \mathit{SITES} \setminus \{N\} \mapsto \textbf{if } v[i] = v[i+1] \textbf{ then } 1 \textbf{ else } 0 \,] \\
t_2 & \triangleq & \text{LEAST } k : \neg\exists i \in \mathit{SITES} \setminus \{0\} : v[i] = (v[0]+k) \bmod (M+1)
\end{array}
$$

Figure 7.11: Definitions added for the proof.

around the ring, and thus a stable configuration will be reached.

This idea can be formally represented by the predicate diagram G_{conc} shown in figure 7.10(b). Like the previous diagram G_{abs}, it does not allow transitions from stable to unstable configurations. Similarly, it rules out moves from unstable configurations with a singular prefix to unstable configurations without a singular prefix. However, the fairness condition of G_{abs} has been replaced with a weaker progress assumption: weak fairness of the system's next-state relation asserts that the protocol should not stop. Also, ordering annotations on transitions have been introduced whose meaning will be explained below.

Definition 7.7 of structural refinement turns out to be liberal enough to allow for such non-local refinement of fairness. Obviously, the refinement mapping f is defined to map the two upper nodes of G_{conc} to the single upper node of G_{abs}, and the lower node of G_{conc} to the lower node of G_{abs}.

Theorem 7.12 *G_{conc} structurally refines G_{abs}.*

Proof. Conditions (1)–(4) of Definition 7.7 hold trivially. In particular, condition (3) is true because G_{conc} does not allow any transitions that are disallowed by G_{abs}, and condition (4) holds because G_{abs} does not contain any ordering annotations that would have to be preserved.

Moreover, there are no actions with strong fairness, and therefore only condition (5) needs to be verified. Now, the ordering annotations (for any well-founded orderings $<_1$ and $<_2$) ensure that any run through G_{conc} must eventually reach the node labelled Stable, and then remain there forever. In particular, the run contains infinitely many nodes whose image disables the $\mathit{Stabilize}$ action, which implies the fairness condition. A run of the modelchecker on the encoding of G_{conc} suffices to prove the condition. Q.E.D.

To conclude the verification of Dijkstra's protocol, it remains to show that G_{conc} conforms to the protocol specification. In particular, we must define the predicates that appear as node labels of G_{conc}, and the terms t_1 and t_2 and their associated well-founded orderings $<_1$ and $<_2$. Figure 7.11 collects the definitions of these predicates and terms. The definitions of $Stable$ and $SignularPrefix$ have been motivated before. The term t_1 records progress that can be attributed to moves of the non-zero processors: it is defined as a bit vector indicating which of the non-zero processors are enabled. Because it is obvious that every move of a non-zero processor disables that processor and does not affect the enabledness of its neighbors to the left, the corresponding well-founded ordering $<_1$ is chosen as the lexicographic ordering on bit vectors of length N. The term t_2, which records progress brought about by moves of processor 0, is defined as the "distance" to the (cyclically) smallest value that does not occur among the register values of processors that are part of the singular prefix. As we have motivated before, a configuration with singular prefix is reached when that distance becomes zero, and therefore $<_2$ is simply chosen as the standard ordering on natural numbers.

Theorem 7.13 *G_{conc} conforms to formula Spec of figure 7.9.*

Proof. We consider the conditions of theorem 7.3. Condition (1) holds because the disjunction of the labels of initial states is trivial. For condition (2), it has to be shown that all actions preserve stability and the existence of a singular prefix, respectively. This is proven by considering the possible moves as defined by formula $Next$ of figure 7.9: every move disables the processor that made the move, it may enable its right-hand neighbor, and does not affect the enabledness of any other processor.

Establishing condition (3) is harder. It follows from the following observations:

- Every step of a non-zero processor i disables itself but does not affect the enabledness of processors j for $0 < j < i$. Therefore, such steps decrease t_1.

- Steps of non-zero processors do not increase the value of t_2: the set $\{v'[1], \ldots, v'[N]\}$ of register values after the step is a subset of the set $\{v[1], \ldots, v[N]\}$ of values before the step. Therefore, if $(v[0] + k) \bmod (M + 1)$ did not occur in the set of register values before the step, it still does not occur in the set of register values thereafter, and the smallest such value k could only have decreased.

- Every action of processor 0 from an unstable state without a singular prefix decreases the value of t_2: if there is no singular prefix, the value $v[0]$ must occur as $v[i+1]$, for some i, and therefore t_2 must be positive. Now, an action of processor 0 increments $v[0]$ but leaves all other registers unchanged, hence t_2 decreases by one.

Condition (4a) is obvious and condition (4b) is vacuously true. Condition (4c) holds because some processor is enabled in all states, as we have argued before. Condition (4d) is a consequence of condition (2) that was already proven, because only one action occurs in the diagram. Overall, the conformance proof has been mechanized in Isabelle/HOL [83] and has required approximately 300 interactions. We found the proofs that rely on modulus arithmetic to be the hardest to carry out in Isabelle, because only little automation was provided. Q.E.D.

7.6 Related Work

There have been many suggestions for visual notations that support the formal development and analysis of systems. Owicki and Lamport [81] defined proof lattices to structure proofs of concurrent programs. A proof lattice is an acyclic finite graph with input assertion P and output assertion Q; it represents the proof of a liveness property $P \leadsto Q$. A structuring mechanism was sketched to introduce or remove invariants. Anderson's implementation of the UNITY logic in the interactive theorem prover HOL [13] makes use of proof lattices. Radhia Cousot [31] introduced proof charts to reason about parallel programs. Termination proofs are based on annotations that identify well-founded orderings. Lamport [63] defines predicate-action diagrams that represent TLA specifications, restricted to safety properties. His diagrams are intended to communicate aspects of specifications; diagrammatic verification is not considered.

Work by Manna et al [33, 68] in the context of the STeP verification system is most closely related to our use of diagrams. Our definitions differ in technical details, but share the same overall motivations. For example, our ordering annotations are local to edges of the diagram, which simplifies the presentation and reduces the number of proof obligations. On the other hand, our fairness annotations are global in that they may concern several edges. Refinement transformations of diagrams such as node splitting and transition removal are considered in [33]. In contrast, our definitions of refinement are not restricted to isolated modifications applied to single nodes or edges, but support non-trivial refinements of fairness and liveness properties.

Probably closest in spirit, although different in detail, to our work is the work by Bensalem, Saïdi et al on the generation of invariants that is incorporated in the InVest and SAL tools [18, 19, 89]. While our approach focuses more on abstract evaluation, they use the theorem prover PVS as their primary workhorse. Another conceptual difference is our emphasis on liveness properties; indeed, we believe that it is here that abstraction-based techniques can be used to full advantage because the deductive verification of liveness properties is inherently tedious and difficult.

There has been much interest in applying Boolean abstractions to software model checking, and several projects have developed special-purpose tools [30, 43, 46] that incorporate techniques to compute abstractions. Within the context of the SAL project, special-purpose abstraction techniques for parameterized systems have been developed [17].

Our style of top-down development has been influenced by the B approach to system development [9, 10, 11]. However, working in the more general setting of temporal logic, we prefer to state abstract fairness requirements early in the development; they can be implemented by well-founded orderings later on.

Chapter 8

Summary and Future Work

We have studied different aspects that arise in the specification and verification of reactive systems in a temporal logic framework. Chapters 2–4 are of mainly theoretical interest as they concern questions of axiomatizability and decidability. The main open question in this area is to prove a relative completeness theorem for first-order TLA* (or similar logics). Partial answers to this question exist in the literature [2, 56, 67], but it is not clear how to systematically extend these results. The theory of ω-regular languages that are invariant under stuttering equivalence of chapter 3 may be of independent interest. It would also be interesting to investigate a class of alternating automata or logical games that correspond to TLA*. This would allow to better separate the logical aspects from the combinatorics involved in deciding emptiness.

The justification of assumption-commitment reasoning from basic logical principles that we have presented in chapter 5 appears to capture the essence of composition rules found in various formalisms. When the abstract rule is applied to concrete formalisms, additional detail reflecting the particular model of computation can be made use of to prove more specific composition rules. One question of interest is to integrate compositional reasoning with hiding and to study the effects of quantification at the level of the abstract logic. Concerning practical applications, assumption-commitment specifications hold great promise for the specification of reusable components. Presently, specifications of such components are rare, and open-system specification still appears to be a theory waiting to be put to use.

In our opinion, the work presented in chapters 6 and 7 holds the biggest immediate promise for the application of formal methods. There are two main streams of ongoing work: on a more theoretical level, the completeness of the approach relative to traditional verification rules should be established. This should at least be possible for the fragment of TLA* without quantification over flexible variables; it remains to be seen how history and possibly prophecy variables can be represented at the levels of diagrams and diagram refinements. On the practical level, we are developing a prototypical implementation of the ideas presented in chapter 7. The main difficulty lies in the integration of existing tools as diverse as model checkers, rewriting engines, and interactive theorem provers. For practical applications, pre-developed abstraction libraries will have to be provided, and special-purpose abstraction methods should be implemented, for example for computing abstractions of parameterized systems.

Bibliography

[1] Martín Abadi. An axiomatization of Lamport's Temporal Logic of Actions. In Jos C. M. Baeten and Jan W. Klop, editors, *CONCUR '90, Theories of Concurrency: Unification and Extension*, volume 458 of *Lecture Notes in Computer Science*, pages 57–69, Berlin, 1990. Springer-Verlag.

[2] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 81(2):253–284, May 1991.

[3] Martín Abadi and Leslie Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, 1993.

[4] Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, May 1995.

[5] Martín Abadi, Leslie Lamport, and Stephan Merz. Refining specifications. Palo Alto, 1994.

[6] Martín Abadi and Stephan Merz. An abstract account of composition. In Jiří Wiedermann and Petr Hajek, editors, *Mathematical Foundations of Computer Science*, volume 969 of *Lecture Notes in Computer Science*, pages 499–508, Berlin, 1995. Springer-Verlag.

[7] Martín Abadi and Stephan Merz. On TLA as a logic. In Manfred Broy, editor, *Deductive Program Design*, NATO ASI series F, pages 235–272. Springer-Verlag, Berlin, 1996.

[8] Martín Abadi and Gordon Plotkin. A logical view of composition. *Theoretical Computer Science*, 114(1):3–30, June 1993.

[9] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[10] J.-R. Abrial. Extending B without changing it (for developing distributed systems). In H. Habrias, editor, *1st Conference on the B method*, pages 169–190. IRIN Institut de recherche en informatique de Nantes, 1996.

[11] J.-R. Abrial. Event-driven sequential programs. http://www-sop.inria.fr/EJC2000/edspc.V12.ps.gz, March 2000.

[12] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.

[13] F. Anderson. *A theorem prover for* UNITY *in Higher Order Logic*. PhD thesis, Technical University of Denmark, 1992.

[14] R. Back and J. von Wright. *Refinement calculus—A systematic introduction*. Springer-Verlag, 1998.

[15] B. Banieqbal and H. Barringer. Temporal logic with fixed points. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, volume 398 of *Lecture Notes in Computer Science*, pages 62–74. Springer-Verlag, 1987.

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

[17] K. Baukus, S. Bensalem, Y. Lakhnech, and K. Stahl. Abstracting WS1S systems to verify parameterized networks. In S. Graf and M. Schwartzbach, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000)*, volume 1785 of *Lecture Notes in Computer Science*, pages 188–203. Springer-Verlag, 2000.

[18] S. Bensalem, Y. Lakhnech, and H. Saidi. Powerful techniques for the automatic generation of invariants. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification: 8th International Conference*, volume 1102 of *Lecture Notes in Computer Science*, pages 323–335. Springer-Verlag, 1996.

[19] Saddek Bensalem, Vijay Ganesh, Yassine Lakhnech, César Mu noz, Sam Owre, Harald Rueß, John Rushby, Vlad Rusu, Hassen Saïdi, N. Shankar, Eli Singerman, and Ashish Tiwari. An overview of SAL. In C. Michael Holloway, editor, *LFM 2000: Fifth NASA Langley Formal Methods Workshop*, pages 187–196, Hampton, VA, jun 2000. NASA Langley Research Center.

[20] Jonathan Bowen and Victoria Stavridou. Safety-critical systems, formal methods, and standards. *BCS/IEE Software Engineering Journal*, 8(4), 1993.

[21] Robert S. Boyer. *A Computational Logic Handbook*. Academic Press, London, 2nd edition edition, 1998.

[22] Manfred Broy. A functional rephrasing of the assumption/commitment specification style. SFB-Bericht 342/10/94, TUM-I9417, Techn. Univ. München, Munich, April 1994.

[23] Manfred Broy, Stephan Merz, and Katharina Spies, editors. *Formal System Specification: The RPC-Memory Specification Case Study*, volume 1169 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[24] Manfred Broy and Ketil Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer-Verlag, Berlin, 2000.

[25] J. R. Büchi. On a decision method in restricted second-order arithmetics. In *International Congress on Logic, Method and Philosophy of Science*, pages 1–12. Stanford University Press, 1962.

[26] Dominique Cansell, Dominique Méry, and Stephan Merz. Predicate diagrams for the verification of reactive systems. In *2nd Intl. Conf. on Integrated Formal Methods (IFM 2000)*, volume 1945 of *Lecture Notes in Computer Science*, pages 380–397, Dagstuhl, Germany, November 2000. Springer-Verlag.

[27] Antonio Cau and Pierre Collette. Parallel composition of assumption-commitment specifications: a unifying approach for shared variable and distributed message passing concurrency. *Acta Informatica*, 1995. To appear.

[28] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, September 1994.

[29] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999.

[30] James C. Corbett, Matthew Dwyer, John Hatcliff, Corina Pasareanu, Robby, Shawn Laubach, and Hongjun Zheng. Bandera: Extracting finite-state models from Java source code. In *22nd International Conference on Software Engineering*, pages 439–448, New York, 2000. ACM Press.

[31] Radhia Cousot. *Fondements de méthodes de preuve d'invariance et de fatalité de programmes parallèles*. PhD thesis, INPL, 1985.

[32] Dennis Dams, Orna Grumberg, and Rob Gerth. Abstract interpretation of reactive systems: Abstractions preserving $\forall$CTL*, $\exists$CTL* and CTL*. In Ernst-Rüdiger Olderog, editor, *Programming Concepts, Methods, and Calculi (PROCOMET '94)*, pages 561–581, Amsterdam, 1994. North Holland/Elsevier.

[33] Luca de Alfaro, Zohar Manna, Henny B. Sipma, and Tomás Uribe. Visual verification of reactive systems. In Ed Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97)*, volume 1217 of *Lecture Notes in Computer Science*, pages 334–350. Springer-Verlag, 1997.

[34] W.-P. de Roever, H. Langmaack, and A. Pnueli, editors. *Compositionality: The Significant Difference*, volume 1536 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[35] D. Detlefs, G. Nelson, and J. Saxe. Simplify: the ESC theorem prover. Technical report, Systems Research Center, Digital Equipment Corporation, Palo Alto, CA, November 1996.

[36] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, November 1974.

[37] Edsger W. Dijkstra. A belated proof of self-stabilization. *Distributed Computing*, 1:5–6, 1986.

[38] E. Allen Emerson. *Handbook of theoretical computer science*, chapter Temporal and modal logic, pages 997–1071. Elsevier Science Publishers B.V., 1990.

[39] Dov Gabbay, Amir Pnueli, S. Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages*, pages 163–173. ACM, 1980.

[40] Susanne Graf and Hassan Saidi. Construction of abstract state graphs with PVS. In Orna Grumberg, editor, *Computer Aided Verification: 9th International Conference*, volume 1254 of *Lecture Notes in Computer Science*, pages 72–83. Springer-Verlag, 1997.

[41] Orna Grumberg and David E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, May 1994.

[42] David Harel and Amir Pnueli. On the development of reactive systems. In K. R. Apt, editor, *Logics and Models of Concurrent Systems*, volume F13 of *NATO ASI Series*, pages 477–498. Springer-Verlag, 1985.

[43] Klaus Havelund and Tom Pressburger. Model checking java programs using java pathfinder. *International Journal on Software Tools for Technology Transfer*, 2(4), 2000.

[44] Hermann Hellwagner. Scalable readers/writers synchronization on shared-memory machines. Technical report, Siemens AG, ZFE ST SN 2, 1993.

[45] Gerard Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, New Jersey, 1991.

[46] Gerard J. Holzmann and Margaret H. Smith. Automating software feature verification. *Bell Labs Technical Journal*, page April, 2000.

[47] Cliff B. Jones. Tentative steps toward a development method for interfering programs. *ACM Transactions on Programming Languages and Systems*, 5(4):596–619, October 1983.

[48] Bengt Jonsson. Simulations between specifications of distributed systems. In Jos C. M. Baeten and Jan Frisco Groote, editors, *CONCUR '91, 2nd International Conference on Concurrency Theory*, volume 527 of *Lecture Notes in Computer Science*, pages 346–360, Amsterdam, The Netherlands, 1991. Springer.

[49] Bengt Jonsson and Yih-Kuen Tsay. Assumption/guarantee specifications in linear-time temporal logic. In P.D. Mosses, M. Nielsen, and M.I. Schwartzbach, editors, *TAPSOFT'95*, volume 915 of *Lecture Notes in Computer Science*, pages 262–276. Springer-Verlag, 1995.

[50] Bernhard Josko. Verifying the correctness of AADL modules using model checking. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, volume 430 of *Lecture Notes in Computer Science*, pages 386–400. Springer-Verlag, Berlin, 1989.

[51] Bernhard Josko. *Modular Specification and Verification of Reactive Systems*. PhD thesis, Univ. Oldenburg, Fachbereich Informatik, April 1993.

[52] H. W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, Univ. of California at Los Angeles, 1968.

[53] Y. Kesten and A. Pnueli. Modularization and abstraction: The keys to practical formal verification. In *23rd Intl. Symp. Mathematical Foundations of Computer Science (MFCS'98)*, volume 1450 of *Lecture Notes in Computer Science*, pages 54–71. Springer-Verlag, 1998.

[54] Yomit Kesten and Amir Pnueli. A complete proof system for QPTL. In Dexter Kozen, editor, *Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 2–12, San Diego, 1995. IEEE Press.

[55] Yonit Kesten and Amir Pnueli. Verifying liveness by augmented abstraction. In *Annual Conference of the European Association for Computer Science Logic (CSL'99)*, Lecture Notes in Computer Science, Madrid, 1999. Springer-Verlag.

[56] N. Klarlund and Fred B. Schneider. Proving nondeterministically specified safety properties using progress measures. *Information and Computation*, 107(1):151–170, November 1993.

[57] Fred Kröger. *Temporal Logic of Programs*, volume 8 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1987.

[58] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not so weak. In *5th Israeli Symposium on Theory of Computing and Systems*, pages 147–158. IEEE Press, 1997.

[59] Simon S. Lam and A. Udaya Shankar. Protocol verification via projections. *IEEE Transactions on Software Engineering*, SE-10(4):325–342, July 1984.

[60] Leslie Lamport. A new solution of Dijkstra's concurrent programming problem. *Communications of the ACM*, 17(8):453–454, 1974.

[61] Leslie Lamport. What good is temporal logic? In R. E. A. Mason, editor, *Information Processing 83: Proceedings of the IFIP 9th World Congress*, pages 657–668, Paris, September 1983. IFIP, North-Holland.

[62] Leslie Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[63] Leslie Lamport. TLA in Pictures. *IEEE Transactions on Software Engineering*, 21(9):768–775, September 1995.

[64] Leslie Lamport. Composition: A way to make proofs harder. In W.-P. de Roever et al, editor, *Compositionality: The Significant Difference*, volume 1536 of *Lecture Notes in Computer Science*, pages 402–423, Berlin, 1998. Springer-Verlag.

[65] Nancy G. Leveson. *System Safety and Computers*. Addison-Wesley, 1995.

[66] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA, 1996.

[67] Nancy Lynch and Frits Vaandrager. Forward and backward simulations. Part I: Untimed systems. Technical Report CS-R9313, CWI, March 1993.

[68] Z. Manna, A. Browne, H.B. Sipma, and T.E. Uribe. Visual abstractions for temporal verification. In A. Haeberer, editor, *AMAST'98*, volume 1548 of *Lecture Notes in Computer Science*, pages 28–41. Springer-Verlag, 1998.

[69] Zohar Manna, Anuchit Anuchitanukul, Nikolaj Bjørner, Anca Browne, Edward Chang, Michael Colón, Luca de Alfaro, Harish Devarajan, Henny Sipma, and Tomás Uribe. STeP: the Stanford temporal prover. Technical Report STAN-CS-TR-94-1518, Department of Computer Science, Stanford University, June 1994.

[70] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems—Specification*. Springer-Verlag, New York, 1992.

[71] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems—Safety properties*. Springer-Verlag, New York, 1995.

[72] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

[73] Stephan Merz. Decidability and incompleteness results for first-order temporal logics of linear time. *Journal of Applied Non-Classical Logic*, 2(2), 1992.

[74] Stephan Merz. Isabelle/TLA. Available on the WWW at `http://www.pst.informatik.uni-muenchen.de/~merz/isabelle/index.html`, 1997. Revised 1999.

[75] Stephan Merz. Model checking: A tutorial overview. In F. Cassez et al., editor, *Modeling and Verification of Parallel Processes*, volume 2067 of *Lecture Notes in Computer Science*, pages 3–38. Springer-Verlag, Berlin, 2001.

[76] A.R. Meyer and L.J. Stockmeyer. Non-elementary word problems in automata theory and logic. In *Proceedings of the AMS Symposium on Complexity of Computation*, 1973.

[77] M. Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, 1988.

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

[79] D. E. Muller. Infinite sequences and finite machines. In *Switching Circuit Theory and Logical Design: Fourth Annual Symposium*, pages 3–16, New York, 1963. IEEE Press.

[80] D.E. Muller, A. Saoudi, and P.E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *3rd IEEE Symposium on Logic in Computer Science*, pages 422–427. IEEE Press, 1988.

[81] Susan Owicki and Leslie Lamport. Proving liveness properties of concurrent programs. *ACM Transactions on Programming Languages and Systems*, 4(3):455–495, July 1982.

[82] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.

[83] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, 1994. See also the Isabelle home page at `http://isabelle.in.tum.de/`.

[84] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on the Foundations of Computer Science*, pages 46–57. IEEE, November 1977.

[85] Amir Pnueli. In transition from global to modular temporal reasoning about programs. In K. R. Apt, editor, *Logics and Models of Concurrent Systems*, volume F 13 of *ASI*, pages 123–144. Springer-Verlag, Berlin, 1985.

[86] Shaz Qadeer and Natarajan Shankar. Verifying a self-stabilizing mutual exclusion algorithm. In David Gries and Willem-Paul de Roever, editors, *Programming Concepts and Methods*, pages 424–443, Shelter Island, N.Y., June 1998. Chapman & Hall.

[87] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[88] Shmuel Safra. On the complexity of ω-automata. In *29th IEEE Symposium on Foundations of Computer Science*, pages 319–327. IEEE Press, 1988.

[89] Hassan Saïdi and Natarajan Shankar. Abstract and model check while you prove. In N. Halbwachs and D. Peled, editors, *11th International Conference on Computer-Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 443–454. Springer-Verlag, 1999.

[90] N. Shankar. Lazy compositional verification. In de Roever et al. [34], pages 541–564.

[91] A. P. Sistla. On verifying that a concurrent program satisfies a nondeterministic specification. Technical Report TR 88-378.01.1, Computer and Intelligent Systems Laboratory, GTE Laboratories, May 1988.

[92] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[93] M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, 1992.

[94] Colin Stirling. *Modal and Temporal Properties of Processes*. Springer-Verlag, Berlin, 2001.

[95] Ketil Stølen, Frank Dederichs, and Rainer Weber. Assumption/commitment rules for networks of asynchronously communicating agents. SFB-Bericht 342/2/93, TUM-I9303, Techn. Univ. München, Munich, February 1993.

[96] Andrzej Szalas. Concerning the semantic consequence relation in first-order temporal logic. *Theoretical Computer Science*, 47:329–334, 1986.

[97] Andrzej Szalas. A complete axiomatic characterization of first-order temporal logic of linear time. *Theoretical Computer Science*, pages 199–214, 1987.

[98] Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, volume B: Formal Models and Semantics*, pages 133–194. Elsevier, Amsterdam, 1990.

[99] Wolfgang Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer-Verlag, New York, 1997.

[100] Wolfgang Thomas. Complementation of Büchi automata revisited. In J. Karhumäki, editor, *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 109–122. Springer-Verlag, 2000.

[101] A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics: An Introduction*, volume 1. North Holland, Amsterdam, 1988.

[102] Moshe Vardi. On the complexity of modular model checking. In *Proceedings of the Tenth Symposium on Logic in Computer Science*. IEEE, June 1995.

[103] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[104] Pierre Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–93, 1983.

[105] Qiwen Xu, Antonio Cau, and Pierre Collette. On unifying assumption-commitment style proof rules for concurrency. In Bengt Jonsson and Joachim Parrow, editors, *Proceedings of the Fifth International Conference on Concurrency Theory (CONCUR '94)*, volume 836 of *Lecture Notes in Computer Science*, pages 267–282, Berlin, 1994. Springer-Verlag.

Auxiliary derivations and proofs

This appendix contains some supplementary material concerning the axiomatizations of propositional GTLA (cf. sections 2.3 and 2.5).

A.1 Derivations in Σ_{TLA^*}

The following derivations are presented using a conventional Hilbert style. Steps based on propositional reasoning are indicated by (prop).

(alw) $\dfrac{\vdash F}{\vdash \Box F}$

(1)	$\vdash F$	(assumption)
(2)	$\mid\!\sim \circ F$	(1)(nex)
(3)	$\mid\!\sim F \to \circ F$	(prop)(2)
(4)	$\vdash \Box[F \to \circ F]_F$	(sq)(3)(prop)
(5)	$\vdash F \to \Box F$	(4)(ax3)(mp)
(6)	$\vdash \Box F$	(mp)(1)(5)

Two applications of (alw) show $F \vdash \Box\Box F$, and the deduction theorem 2.7 implies $\vdash \Box F \to \Box\Box F$. Together with (ax1) we obtain theorem

(T1) $\vdash \Box F \leftrightarrow \Box\Box F$

(T2) $\vdash \Box[A]_v \leftrightarrow \Box\Box[A]_v$

(1)	$\mid\!\sim \Box[A]_v \to \circ\Box[A]_v$	(pax4)
(2)	$\vdash \Box[\Box[A]_v \to \circ\Box[A]_v]_v$	(sq)(1)
(3)	$\vdash \Box[A]_v \to \Box\Box[A]_v$	(2)(ax3)(mp)
(4)	$\vdash \Box\Box[A]_v \to \Box[A]_v$	(ax1)
(5)	$\vdash \Box[A]_v \leftrightarrow \Box\Box[A]_v$	(prop)(3)(4)

(T3) $\vdash \Box[[A]_v]_v \leftrightarrow \Box[A]_v$

(1)	$\mid\!\sim A \rightarrow [A]_v$	(pax0)
(2)	$\vdash \Box[A \rightarrow [A]_v]_v$	(sq)(1)
(3)	$\vdash \Box[A]_v \rightarrow \Box[[A]_v]_v$	(3)(ax4)(mp)
(4)	$\mid\!\sim [A]_v \rightarrow (v' \neq v \rightarrow A)$	(pax0)
(5)	$\vdash \Box[[A]_v]_v \rightarrow (\Box[v' \neq v]_v \rightarrow \Box[A]_v)$	(4)(ax4)(prop)
(6)	$\vdash \Box[v' \neq v]_v$	(ax5)
(7)	$\vdash \Box[[A]_v]_v \rightarrow \Box[A]_v$	(prop)(5)(6)
(8)	$\vdash \Box[[A]_v]_v \leftrightarrow \Box[A]_v$	(prop)(3)(7)

(T4) $\vdash \Box[A]_v \rightarrow \Box[[A]_v]_w$

(1)	$\vdash \Box[A]_v \rightarrow \Box\Box[A]_v$	(T2)(prop)
(2)	$\vdash \Box\Box[A]_v \rightarrow \Box[\Box\Box[A]_v]_w$	(ax2)
(3)	$\mid\!\sim \Box\Box[A]_v \rightarrow \Box[A]_v$	(ax1)(pre)
(4)	$\mid\!\sim \Box[A]_v \rightarrow [A]_v$	(pax4)
(5)	$\vdash \Box[\Box\Box[A]_v \rightarrow [A]_v]_w$	(3)(4)(sq)
(6)	$\vdash \Box[\Box\Box[A]_v]_w \rightarrow \Box[[A]_v]_w$	(5)(ax4)(mp)
(7)	$\vdash \Box[A]_v \rightarrow \Box[[A]_v]_w$	(prop)(1)(2)(5)

(T5) $\vdash \Box[[A]_w]_v \rightarrow \Box[[A]_v]_w$

(1)	$\vdash \Box[[A]_w]_v \rightarrow \Box[[[A]_w]_v]_w$	(T4)
(2)	$\mid\!\sim [[A]_w]_v \rightarrow [[A]_v]_w$	(pax0)
(3)	$\vdash \Box[[[A]_w]_v \rightarrow [[A]_v]_w]_w$	(sq)(2)
(4)	$\vdash \Box[[[A]_w]_v]_w \rightarrow \Box[[[A]_v]_w]_w$	(3)(ax4)(mp)
(5)	$\vdash \Box[[[A]_v]_w]_w \rightarrow \Box[[A]_v]_w$	(T3)(prop)
(6)	$\vdash \Box[[A]_w]_v \rightarrow \Box[[A]_v]_w$	(prop)(1)(4)(5)

(T6) $\vdash \Box F \rightarrow \Box[\circ F]_v$

(1)	$\vdash \Box F \rightarrow \Box[\Box F]_v$	(ax2)
(2)	$\mid\!\sim \Box F \rightarrow \circ\Box F$	(pax3)
(3)	$\vdash \Box F \rightarrow F$	(ax1)
(4)	$\mid\!\sim \circ(\Box F \rightarrow F)$	(nex)(3)
(5)	$\mid\!\sim \circ\Box F \rightarrow \circ F$	(4)(pax2)(mp)
(6)	$\mid\!\sim \Box F \rightarrow \circ F$	(prop)(2)(5)
(7)	$\vdash \Box[\Box F \rightarrow \circ F]_v$	(sq)(6)
(8)	$\vdash \Box[\Box F]_v \rightarrow \Box[\circ F]_v$	(7)(ax4)(mp)
(9)	$\vdash \Box F \rightarrow \Box[\circ F]_v$	(prop)(1)(8)

(T7) $\mid\!\sim \Box F \leftrightarrow F \wedge \circ\Box F$

(1)	$\vdash \Box F \rightarrow F$	(ax1)
(2)	$\mid\!\sim \Box F \rightarrow F$	(1)(pre)
(3)	$\mid\!\sim \Box F \rightarrow \circ\Box F$	(pax3)
(4)	$\mid\!\sim \Box F \rightarrow F \wedge \circ\Box F$	(prop)(2)(3)
(5)	$\mid\!\sim \circ\Box F \rightarrow \Box[\circ F]_F$	(pax5)(prop)
(6)	$\mid\!\sim \circ F \rightarrow (F \rightarrow \circ F)$	(pax0)
(7)	$\vdash \Box[\circ F \rightarrow (F \rightarrow \circ F)]_F$	(sq)(6)
(8)	$\vdash \Box[\circ F]_F \rightarrow \Box[F \rightarrow \circ F]_F$	(7)(ax4)(prop)
(9)	$\vdash \Box[\circ F]_F \rightarrow (F \rightarrow \Box F)$	(8)(ax3)(mp)
(10)	$\mid\!\sim \circ\Box F \rightarrow (F \rightarrow \Box F)$	(pre)(9)(5)(prop)
(11)	$\mid\!\sim \Box F \leftrightarrow F \wedge \circ\Box F$	(prop)(4)(10)

(T8) $\mid\!\sim \circ(F \wedge G) \leftrightarrow \circ F \wedge \circ G$

(1)	$\vdash F \wedge G \rightarrow F$	(ax0)
(2)	$\mid\!\sim \circ(F \wedge G \rightarrow F)$	(nex)(1)
(3)	$\mid\!\sim \circ(F \wedge G) \rightarrow \circ F$	(pax2)(2)(mp)
(4)	$\mid\!\sim \circ(F \wedge G) \rightarrow \circ G$	(similar)
(5)	$\mid\!\sim \circ(F \rightarrow \neg G) \rightarrow (\circ F \rightarrow \neg\circ G)$	(pax2)(pax1)(prop)
(6)	$\mid\!\sim \circ F \wedge \circ G \rightarrow \circ\neg(F \rightarrow \neg G)$	(prop)(5)(pax1)
(7)	$\vdash \neg(F \rightarrow \neg G) \rightarrow F \wedge G$	(ax0)
(8)	$\mid\!\sim \circ\neg(F \rightarrow \neg G) \rightarrow \circ(F \wedge G)$	(nex)(7)(pax2)(mp)
(9)	$\mid\!\sim \circ(F \wedge G) \leftrightarrow \circ F \wedge \circ G$	(prop)(3)(4)(6)(8)

A.2 Proof of theorem 2.8

We now prove that the relations $\vdash$ and $\mid\!\sim$ agree on formulas. One direction is immediate by rule (pre), the other direction requires a careful examination of possible derivations that result in a statement $\mid\!\sim A$. To make the notation more concise, we use a sequence notation for implications that are nested to the right. For example, we write $\langle A, B\rangle \to C$ for $A \to (B \to C)$.

First, we need to be more precise about (pax0). It is well known that the following axiom schemes, together with modus ponens, completely axiomatize propositional calculus:

(pax0.1)	$\vdash \langle A, B\rangle \to A$
(pax0.2)	$\vdash \langle\langle A, B\rangle \to C, A \to B, A\rangle \to C$
(pax0.3)	$\vdash \langle \neg B \to \neg A, A\rangle \to B$
(pax0.4)	$\vdash \langle B \to \neg A, A\rangle \to \neg B$
(pax0.5)	$\vdash \langle \neg B \to A, \neg A\rangle \to B$
(pax0.6)	$\vdash \langle B \to A, \neg A\rangle \to \neg B$

In fact, (pax0.1)–(pax0.3) are already sufficient for propositional completeness. We have added the redundant axioms (pax0.4)–(pax0.6) in order to prove that, essentially, there is no need for doubly negated pre-formulas in derivations, as shown in lemma A.1 below.

Moreover, we replace (pax1) and (pax4) by the following axioms, which do not contain derived propositional operators. The axioms (pax1.3) and (pax1.4) are easily derived in the original system Σ_{TLA^*}.

(pax1.1)	$\mid\!\sim \circ\neg F \to \neg\circ F$
(pax1.2)	$\mid\!\sim \neg\circ F \to \circ\neg F$
(pax1.3)	$\mid\!\sim \neg\circ\neg F \to \circ F$
(pax1.4)	$\mid\!\sim \circ F \to \neg\circ\neg F$
(pax4.1)	$\mid\!\sim \langle \Box[A]_v, v, \neg\circ v\rangle \to A$
(pax4.2)	$\mid\!\sim \langle \Box[A]_v, \neg v, \circ v\rangle \to A$
(pax4.3)	$\mid\!\sim \Box[A]_v \to \circ\Box[A]_v$
(pax4.4)	$\mid\!\sim \langle\langle v, \neg\circ v\rangle \to A, \langle \neg v, \circ v\rangle \to A, \circ\Box[A]_v\rangle \to \Box[A]_v$

Lemma A.1 *For any derivation $\vdash F$ or $\mid\!\sim A$ where F or A is a (pre-)formula that contains a sub-pre-formula of the form $\neg\neg B$ there is a derivation $\vdash F^*$ or $\mid\!\sim A^*$ of at most the same length as the original derivation where F^* and A^* denote F and A with $\neg\neg B$ replaced by B.*

Proof. Straightforward induction on the length of derivations. In particular, $\vdash F^*$ is an instance of (ax0) iff $\vdash F$ is. The addition of (pax0.4)–(pax0.6), (pax1.3), and (pax1.4) ensure that the assertion holds for every possible instance of (pax0.1)–(pax5). Q.E.D.

The assertions 1 and 4 of the following lemma are used in the proof of theorem 2.8. The remaining assertions are needed for the induction step.

Lemma A.2 *Assume that $\mathcal{F} = \langle F_1, \ldots, F_n\rangle$ and $\mathcal{P} = \langle P_1, \ldots, P_m\rangle$ are sequences of formulas and pre-formulas, for some $n, m \in \mathbb{N}$, that sequence $\mathcal{A} = \langle A_1, \ldots, A_{n+m}\rangle$ is obtained by merging sequences $\mathcal{F}$ and $\mathcal{P}$, and that F, G are formulas and P is a pre-formula. For $i = 1, \ldots, m$, let $\mathcal{F}^i$ denote the prefix of $\mathcal{F}$ that consists of those formulas in $\mathcal{F}$ that precede P_i in $\mathcal{A}$. For $0 \le k \le m$, let $\mathcal{B}^k$ denote the sequence $\langle P_1, \ldots, P_k, \circ F, P_{k+1}, \ldots, P_m\rangle$ and let $\mathcal{C}^k$ denote $\langle P_1, \ldots, P_k, \neg\circ F, P_{k+1}, \ldots, P_m\rangle$.*

1. *If $\mid\!\sim \mathcal{A} \to F$ and $\mid\!\sim \mathcal{F}^i \to P_i$ for $i = 1, \ldots, m$ then $\vdash \mathcal{F} \to F$.*
2. *If $\mid\!\sim \mathcal{A} \to P$, $\mid\!\sim \neg P$, and $\mid\!\sim \mathcal{F}^i \to P_i$ for $i = 1, \ldots, m$ then $\vdash \neg\mathcal{F}$. In particular, if $n = 0$ then $\vdash$* **false**.
3. *If $\mid\!\sim \mathcal{A} \to \neg P$, $\mid\!\sim P$, and $\mid\!\sim \mathcal{F}^i \to P_i$ for $i = 1, \ldots, m$ then $\vdash \neg\mathcal{F}$. In particular, if $n = 0$ then $\vdash$* **false**.
4. *If $\mid\!\sim \mathcal{A} \to \circ F$ and $\mid\!\sim \mathcal{F}^i \to P_i$ for $i = 1, \ldots, m$ then $\vdash \mathcal{F} \to \Box F$.*
5. *If $\mid\!\sim \mathcal{A} \to \neg\circ F$ and $\mid\!\sim \mathcal{F}^i \to P_i$ for $i = 1, \ldots, m$ then $\vdash \mathcal{F} \to \Box\neg F$.*
6. *If $\mid\!\sim \mathcal{B}^k \to G$ and $\mid\!\sim P_1, \ldots, \mid\!\sim P_k, \mid\!\sim \circ F \to P_{k+1}, \ldots, \mid\!\sim \circ F \to P_m$ then $\vdash \neg G \to \Box\neg F$.*
7. *If $\mid\!\sim \mathcal{C}^k \to G$ and $\mid\!\sim P_1, \ldots, \mid\!\sim P_k, \mid\!\sim \neg\circ F \to P_{k+1}, \ldots, \mid\!\sim \neg\circ F \to P_m$ then $\vdash \neg G \to \Box F$.*
8. *If $\mid\!\sim \mathcal{B}^k \to \circ G$, $\mid\!\sim P_1, \ldots, \mid\!\sim P_k, \mid\!\sim \circ F \to P_{k+1}, \ldots, \mid\!\sim \circ F \to P_m$ then $\vdash F \to G$.*
9. *If $\mid\!\sim \mathcal{B}^k \to \neg\circ G$, $\mid\!\sim P_1, \ldots, \mid\!\sim P_k, \mid\!\sim \circ F \to P_{k+1}, \ldots, \mid\!\sim \circ F \to P_m$ then $\vdash F \to \neg G$.*
10. *If $\mid\!\sim \mathcal{C}^k \to \circ G$, $\mid\!\sim P_1, \ldots, \mid\!\sim P_k, \mid\!\sim \neg\circ F \to P_{k+1}, \ldots, \mid\!\sim \neg\circ F \to P_m$ then $\vdash \neg F \to G$.*
11. *If $\mid\!\sim \mathcal{C}^k \to \neg\circ G$, $\mid\!\sim P_1, \ldots, \mid\!\sim P_k, \mid\!\sim \neg\circ F \to P_{k+1}, \ldots, \mid\!\sim \neg\circ F \to P_m$ then $\vdash \neg F \to \neg G$.*
12. *If $\mid\!\sim \mathcal{B}^k \to P$, $\mid\!\sim P_1, \ldots, \mid\!\sim P_k, \mid\!\sim \circ F \to P_{k+1}, \ldots, \mid\!\sim \circ F \to P_m$, and $\mid\!\sim \neg P$ then $\vdash \neg F$.*

13. *If* $\mid\!\sim \mathcal{B}^k \to \neg P$, $\mid\!\sim P_1, \ldots, \mid\!\sim P_k$, $\mid\!\sim \bigcirc F \to P_{k+1}, \ldots, \mid\!\sim \bigcirc F \to P_m$, *and* $\mid\!\sim P$ *then* $\vdash \neg F$.

14. *If* $\mid\!\sim \mathcal{C}^k \to P$, $\mid\!\sim P_1, \ldots, \mid\!\sim P_k$, $\mid\!\sim \neg\bigcirc F \to P_{k+1}, \ldots, \mid\!\sim \neg\bigcirc F \to P_m$, *and* $\mid\!\sim \neg P$ *then* $\vdash F$.

15. *If* $\mid\!\sim \mathcal{C}^k \to \neg P$, $\mid\!\sim P_1, \ldots, \mid\!\sim P_k$, $\mid\!\sim \neg\bigcirc F \to P_{k+1}, \ldots, \mid\!\sim \neg\bigcirc F \to P_m$, *and* $\mid\!\sim P$ *then* $\vdash F$.

Proof. We prove all assertions simultaneously, using induction on the overall complexity of the assumed derivations, which we measure as a pair whose first component is the number of steps and whose second component is the combined length of the derived formulas (i.e., the number of symbols).

1. Consider the last axiom or rule used in the assumed derivation of $\mid\!\sim \mathcal{A} \to F$.

   (pax0.1) : Then $\mathcal{A} = \langle F, A\rangle$, for some A.

   (a) If F is from $\mathcal{P}$ then by assumption we already have $\mid\!\sim F$, and conclude $\vdash F$ by induction hypothesis for assertion 1. If A is also from $\mathcal{P}$, this is exactly what we want, otherwise, the desired conclusion $\vdash A \to F$ follows by propositional logic.

   (b) If F and A are both from $\mathcal{F}$, we have $\vdash \langle F, A\rangle \to F$ by (ax0.1).

   (c) If F is from $\mathcal{F}$ and A is from $\mathcal{P}$, we need to show $\vdash F \to F$, which holds trivially.

   (pax0.2) : Then $\mathcal{A} = \langle\langle A, B\rangle \to F, A \to B, A\rangle$, for some A, B.

   (a) If $\langle A, B\rangle \to F$ is a formula, then A, B, and F are formulas (even though some or all of $\langle A, B\rangle \to F$, $A \to B$ or A may be from $\mathcal{P}$), and we may use the induction hypothesis for assertion 1 to replace "$\mid\!\sim$" by "$\vdash$" in every assumption $\mid\!\sim \mathcal{F}^i \to P_i$, whence the assertion follows propositionally.

   (b) If $\langle A, B\rangle \to F$ is not a formula, neither is $A \to B$ (because F is a formula), so $\langle A, B\rangle \to F$ and $A \to B$ must both be from $\mathcal{P}$, and by assumption we have derivations $\mid\!\sim \langle A, B\rangle \to F$ and $\mid\!\sim A \to B$.

   (b1) If A is from $\mathcal{F}$ then we use the induction hypothesis for assertion 1 (where $\mathcal{F}' = \langle A\rangle$ and $\mathcal{P}' = \langle B\rangle$) to conclude $\vdash A \to F$, which is exactly the desired conclusion.

   (b2) If A is from $\mathcal{P}$ then we also have $\mid\!\sim A$, and apply rule (pmp) to obtain a derivation of the same overall length as the original derivation of the shorter formulas $\mid\!\sim \langle A, B\rangle \to F$, $\mid\!\sim A$, and $\mid\!\sim B$. Hence, we may use the induction hypothesis for assertion 1 (for $\mathcal{F}' = \langle\rangle$, $\mathcal{P}' = \langle A, B\rangle$) to conclude $\vdash F$, as desired.

   (pax0.3) : Then $\mathcal{A} = \langle \neg F \to \neg A, A\rangle$, for some A.

(a) If $\neg F \to \neg A$ and A are both from $\mathcal{F}$ (so $\mathcal{A} = \mathcal{F}$), then $\vdash \mathcal{A} \to F$ is simply an instance of (ax0).

(b) If $\neg F \to \neg A$ is from $\mathcal{P}$ and A is from $\mathcal{F}$, we have $\mid\!\sim \neg F \to \neg A$ by assumption, but $\neg F \to \neg A$ is a formula, and we may use the induction hypothesis for assertion 1 (where $\mathcal{F}' = \mathcal{P}' = \langle\rangle$) to obtain $\vdash \neg F \to \neg A$, hence $\vdash A \to F$ follows propositionally.

(c) If $\neg F \to \neg A$ and A are both from $\mathcal{P}$, then we have $\mid\!\sim \neg F \to \neg A$ and $\mid\!\sim A$, which implies $\vdash \neg\neg F$ by the induction hypothesis for assertion 3, and $\vdash F$ follows by propositional logic.

(d) If $\neg F \to \neg A$ is from $\mathcal{F}$ and A is from $\mathcal{P}$, then A is again a formula, and the induction hypothesis for assertion 1 yields $\vdash (\neg F \to \neg A) \to A$, and therefore $\vdash A$ by propositional logic. Hence, $\vdash (\neg F \to \neg A) \to F$ follows propositionally.

(pax0.4)–(pax0.6) : similar, where (pax0.5) and (pax0.6) depend on assertion 2 instead of 3.

(pax1.1)–(pax1.4), (pax2), (pax3) : not applicable

(pax4.1) : Then $\mathcal{A} = \langle \Box[F]_v, v, \neg\circ v\rangle$, for some atomic proposition v. Clearly, $\neg\circ v$ must be from $\mathcal{P}$.

(a) If $\Box[F]_v$ and v are both from $\mathcal{F}$, we have $\mid\!\sim \langle \Box[F]_v, v\rangle \to \neg\circ v$ by assumption, and the induction hypothesis for assertion 5 lets us conclude $\vdash \langle \Box[F]_v, v\rangle \to \Box\neg v$. Using (ax1) and propositional logic, this implies $\vdash \Box[F]_v \to \neg v$, and the desired conclusion, $\vdash \langle \Box[F]_v, v\rangle \to F$, follows propositionally.

(b) If $\Box[F]_v$ is from $\mathcal{F}$, but v is from $\mathcal{P}$, we have both $\mid\!\sim \Box[F]_v \to v$ and $\mid\!\sim \Box[F]_v \to \neg\circ v$. Using the induction hypotheses for assertions 1 and 5, we obtain $\vdash \Box[F]_v \to v$ and $\vdash \Box[F]_v \to \Box\neg v$. By (ax1) and propositional logic, it follows that $\vdash \neg\Box[F]_v$, hence $\vdash \Box[F]_v \to F$.

(c) The remaining cases are similar.

(pax4.2) : analogous, using assertion 4 instead of 5.

(pax4.3) : not applicable.

(pax4.4) : Then F is $\Box[P]_v$ and $\mathcal{A} = \langle\langle v, \neg\circ v\rangle \to P, \langle\neg v, \circ v\rangle \to P, \circ\Box[P]_v\rangle$, and clearly, all its elements must be from $\mathcal{P}$. In particular, it follows that $\mid\!\sim \circ\Box[P]_v$, and therefore $\vdash \Box\Box[P]_v$, by induction hypothesis for assertion 4. The desired conclusion $\vdash \Box[P]_v$ follows with the help of (ax1).

(pax5) : Then F is $\Box[\circ G]_v$, for some formula G, and $\mathcal{A} = \mathcal{P} = \langle \circ\Box G\rangle$, and by assumption we have $\mid\!\sim \circ\Box G$. Using the induction hypothesis for assertion 4, we conclude $\vdash \Box\Box G$, hence also $\vdash \Box[\circ G]_v$, by axiom (ax1) and (T6).

(pmp) : Then we have $\mid\!\sim P_0$ and $\mid\!\sim \langle P_0, A_1, \ldots, A_{n+m}\rangle \to F$, for some pre-formula P_0. By the induction hypothesis for assertion 1, where $\mathcal{F}' = \mathcal{F}$ and $\mathcal{P}' = \langle P_0, P_1, \ldots, P_m\rangle$, we conclude $\mathcal{F} \to F$ as desired.

(pre) : Then we have $\vdash \mathcal{A} \to F$, and in particular, $P_1, \ldots, P_m$ are formulas, so we may use the induction hypothesis for assertion 1 to conclude $\vdash \mathcal{F}^i \to P_i$ from the assumptions $\mid\!\sim \mathcal{F}^i \to P_i$. Therefore, have $\vdash \mathcal{A} \to F$ (necessary in order to apply rule $\vdash \mathcal{F} \to F$ follows by propositional reasoning.

(nex) : not applicable.

2. Again, consider the last axiom or rule used in the assumed derivation of $\mid\!\sim \mathcal{A} \to P$.

(pax0.1) : Then $\mathcal{A} = \langle P, A \rangle$ for some A.

(a) If P is from $\mathcal{F}$ then P is a formula, and from the assumption $\mid\!\sim \neg P$ and the induction hypothesis for assertion 1 we conclude $\vdash \neg P$. If A is from $\mathcal{P}$ then this is exactly what we need. Otherwise, we may conclude $\vdash \neg(P \wedge A)$ by propositional logic.

(b) If P is from $\mathcal{P}$ then we have both $\mid\!\sim \neg P$ and $\mid\!\sim P$, and we may apply the induction hypothesis for assertion 2 (where $\mathcal{F}' = \mathcal{P}' = \langle\rangle$) to conclude $\vdash$ **false**, which implies the assertion, whether A is from $\mathcal{F}$ or not.

(pax0.2) : Then $\mathcal{A} = \langle \langle A, B \rangle \to P, A \to B, A \rangle$, for some A, B.

(a) If $\langle A, B \rangle \to P$ is a formula then A, B, and P are all formulas, and the induction hypothesis for assertion 1 allows us to replace "$\mid\!\sim$" by "$\vdash$" in every assumption $\mid\!\sim \mathcal{F}^i \to P_i$ and also in $\mid\!\sim \neg P$. The assertion follows by propositional reasoning.

(b) Now suppose that $\langle A, B \rangle \to P$ is not a formula, so it must be from $\mathcal{P}$, and we have $\mid\!\sim \langle A, B \rangle \to P$.

(b1) If $A \to B$ and A are both from $\mathcal{F}$, then A and B are both formulas. From $\mid\!\sim \neg P$ and the induction hypothesis for assertion 2 (choosing $\mathcal{F}' = \langle A, B \rangle$ and $\mathcal{P}' = \langle\rangle$) conclude $\vdash \neg(A \wedge B)$, which implies the assertion $\vdash \neg((A \to B) \wedge A)$ by propositional logic.

(b2) If $A \to B$ is from $\mathcal{F}$ and A is from $\mathcal{P}$, then A and B are again both formulas. By assumption, we have $\mid\!\sim (A \to B) \to A$, and therefore also $\vdash (A \to B) \to A$, by the induction hypothesis for assertion 1. By propositional reasoning, we conclude $\vdash A$. From the assumptions $\mid\!\sim \langle A, B \rangle \to P$ and $\mid\!\sim \neg P$, we may conclude $\vdash \neg(A \wedge B)$ by the induction hypothesis for assertion 2. Together, $\vdash A$ and $\vdash \neg(A \wedge B)$ imply $\vdash \neg(A \to B)$, by propositional logic.

(b3) If $A \to B$ is from $\mathcal{P}$ and A is from $\mathcal{F}$, we have $\mid\!\sim \langle A, B \rangle \to P$ as well as $\mid\!\sim A \to B$ and $\mid\!\sim \neg P$. Using the induction hypothesis for assertion 2 (where $\mathcal{F}' = \langle A \rangle$ and $\mathcal{P}' = \langle B \rangle$) we conclude $\vdash \neg A$, which is the assertion.

(b4) If $A \to B$ and A are both from $\mathcal{P}$, we have $\mid\!\sim A \to B$ and $\mid\!\sim A$. By rule (pmp), we obtain a derivation of the same length of the simpler formulas $\mid\!\sim A$, $\mid\!\sim B$, $\mid\!\sim \langle A, B \rangle \to P$, and $\mid\!\sim \neg P$. The induction hypothesis for assertion 2 gives the desired conclusion, $\vdash$ **false**.

(pax0.3) : Then $\mathcal{A} = \langle \neg P \to \neg A, A \rangle$, for some A.

(a) If $\neg P \to \neg A$ and A are both from $\mathcal{F}$ then, in particular, $\neg P$ is a formula, and we may conclude $\vdash \neg P$ from the assumption $\mid\!\sim \neg P$ and the induction hypothesis for assertion 1. The assertion $\vdash \neg((\neg P \to \neg A) \wedge A)$ follows by propositional logic.

(b) If $\neg P \to \neg A$ is from $\mathcal{F}$ and A is from $\mathcal{P}$ then again we have $\vdash \neg P$ by the induction hypothesis for assertion 1. Moreover, we have the assumption $\mid\!\sim (\neg P \to \neg A) \to A$, and we may again apply the induction hypothesis for assertion 1 to conclude $\vdash (\neg P \to \neg A) \to A$, from which $\vdash A$ follows by propositional logic. Together, $\vdash \neg P$ and $\vdash A$ imply $\vdash \neg(\neg P \to \neg A)$ as desired.

(c) If $\neg P \to \neg A$ is from $\mathcal{P}$ and A is from $\mathcal{F}$, we have $\mid\!\sim \neg P \to \neg A$ and $\mid\!\sim \neg P$, hence we may apply (pmp) to obtain a derivation of the same length as the assumed one of the simpler formula $\mid\!\sim \neg A$, which by the induction hypothesis for assertion 1 implies $\vdash \neg A$, as desired.

(d) If $\neg P \to \neg A$ and A are both from $\mathcal{P}$, we have $\mid\!\sim \neg P$, $\mid\!\sim \neg P \to \neg A$, $\mid\!\sim A$ and may conclude the assertion $\vdash$ **false** by the induction hypothesis for assertion 3 (where $\mathcal{F}' = \langle\rangle$, $\mathcal{P}' = \langle \neg P \rangle$, and P' is A).

(pax0.4)–(pax0.6) : similar; we only give the proof for (pax0.5), which relies on lemma A.1. So $\mathcal{A} = \langle B \to \neg A, A \rangle$ and P is $\neg B$, for some B.

(a) If $B \to \neg A$ and A are both from $\mathcal{F}$, from the assumption $\mid\!\sim \neg\neg B$ and the induction hypothesis for assertion 1 we conclude $\vdash \neg\neg B$. The assertion $\vdash \neg((B \to \neg A) \wedge A)$ follows by propositional logic.

(b) If $B \to \neg A$ is from $\mathcal{F}$ and A is from $\mathcal{P}$ then again we have $\vdash \neg\neg B$ by the induction hypothesis for assertion 1. Moreover, we have the assumption $\mid\!\sim (B \to \neg A) \to A$, and we may again apply the induction hypothesis for assertion 1 to conclude $\vdash (B \to \neg A) \to A$, from which $\vdash A$ follows by propositional logic. Together, $\vdash \neg\neg B$ and $\vdash A$ imply $\vdash \neg(B \to \neg A)$ as desired.

(c) If $B \to \neg A$ is from $\mathcal{P}$ and A is from $\mathcal{F}$, we have $\mid\!\sim B \to \neg A$ and $\mid\!\sim \neg\neg B$. By lemma A.1 there is a derivation $\mid\!\sim B$ whose length does not exceed that of the derivation of $\mid\!\sim \neg\neg B$. Using rule (pmp), we therefore obtain a derivation of at most the same length as the assumed one of the simpler formula $\mid\!\sim \neg A$, which by the induction hypothesis for assertion 1 implies $\vdash \neg A$, as desired.

(d) If $B \to \neg A$ and A are both from $\mathcal{P}$, we have $\mid\!\sim \neg\neg B$, $\mid\!\sim B \to \neg A$, and $\mid\!\sim A$. Again, by lemma A.1 there is a derivation of $\mid\!\sim B$ whose length does not exceed that of the derivation of $\mid\!\sim \neg\neg B$. Therefore, we obtain the assertion $\vdash$ **false** by the induction hypothesis for assertion 3 (where $\mathcal{F}' = \langle\rangle$, $\mathcal{P}' = \langle \neg P \rangle$, and P' is A).

(pax1.1) : Then $\mathcal{A} = \mathcal{P} = \langle \circ\neg F \rangle$ and P is $\circ\neg F$. By assumption, we have $\mid\!\sim \neg\circ F$, and therefore also $\vdash \Box\neg F$, using the induction hypothesis for assertion 4. On the other hand, we have $\mid\!\sim \neg\neg\circ F$. By lemma A.1, there is a derivation $\mid\!\sim \circ F$ whose length does not exceed that of $\mid\!\sim \neg\neg\circ F$, hence we may conclude $\vdash \Box F$, by the induction hypothesis for assertion 4. Together, $\vdash \Box\neg F$ and $\vdash \Box F$ imply $\vdash$ **false** with the help of (ax1).

(pax1.2) : Then $\mathcal{A} = \mathcal{P} = \langle \neg\circ F\rangle$ and P is $\circ\neg F$. We have both $\mid\!\sim \neg\circ F$ and $\mid\!\sim \neg\circ\neg F$, so we conclude $\vdash \Box\neg F$ and $\vdash \Box\neg\neg F$, by the induction hypothesis for assertion 5. Again, $\vdash$ **false** follows.

(pax1.3), (pax1.4) : similar.

(pax2) : Then $\mathcal{A} = \mathcal{P} = \langle \circ(F \to G), \circ F\rangle$ and P is $\circ G$, for some formulas F and G. By assumption, we have $\mid\!\sim \circ(F \to G)$, $\mid\!\sim \circ F$, and $\mid\!\sim \neg\circ G$. By the induction hypotheses for assertions 4 and 5 we conclude $\vdash \Box(F \to G)$, $\vdash \Box F$, and $\vdash \Box\neg G$. But the first two imply $\vdash \Box G$, so we have again $\vdash$ **false**.

(pax3) : Then $\mathcal{A} = \langle \Box F\rangle$ and P is $\circ\Box F$, for some formula F. From the assumption $\mid\!\sim \neg\circ\Box F$ and the induction hypothesis for assertion 5 we deduce $\vdash \Box\neg\Box F$ and thus also $\vdash \neg\Box F$, by (ax1).

If $\Box F$ is from $\mathcal{F}$ then this is exactly what we need to prove. If $\Box F$ is from $\mathcal{P}$ then we have in addition $\mid\!\sim \Box F$, and thus $\vdash \Box F$, by the induction hypothesis for assertion 1. Together, we obtain $\vdash$ **false**, as desired.

(pax4.1) : Then $\mathcal{A} = \langle \Box[P]_v, v, \neg\circ v\rangle$, and clearly, $\neg\circ v$ must be from $\mathcal{P}$.

(a) If $\Box[P]_v$ and v are both from $\mathcal{F}$, we have $\mid\!\sim \langle \Box[P]_v, v\rangle \to \neg\circ v$, and the induction hypothesis for assertion 5 implies $\vdash \langle \Box[P]_v, v\rangle \to \Box\neg v$. The assertion $\vdash \neg(\Box[P]_v \wedge v)$ follows with the help of (ax1) and propositional logic.

(b) If $\Box[P]_v$ is from $\mathcal{F}$ and v is from $\mathcal{P}$, we have both $\mid\!\sim \Box[P]_v \to v$ and $\mid\!\sim \Box[P]_v \to \neg\circ v$. Using, respectively, the induction hypotheses for assertions 1 and 4, we conclude $\vdash \Box[P]_v \to v$ and $\vdash \Box[P]_v \to \Box\neg v$. The assertion $\vdash \neg\Box[P]_v$ now follows by axiom (ax1) and propositional logic.

(c) If $\Box[P]_v$ is from $\mathcal{P}$ and v is from $\mathcal{F}$, we have $\mid\!\sim v \to \neg\circ v$, and therefore $\vdash v \to \Box\neg v$, by the induction hypothesis for assertion 5. The assertion $\vdash \neg v$ again follows by (ax1) and propositional logic.

(d) If $\Box[P]_v$ and v are both from $\mathcal{P}$, we have in particular $\mid\!\sim v$ and $\mid\!\sim \neg\circ v$. Using the induction hypotheses for assertions 1 and 5, we conclude $\vdash v$ and $\vdash \Box\neg v$, and $\vdash$ **false** follows with the help of (ax1) and propositional logic.

(pax4.2) : similar.

(pax4.3) : Then $\mathcal{A} = \langle \Box[Q]_v\rangle$ and P is $\circ\Box[Q]_v$, for some pre-formula Q and atomic proposition v. The assumption $\mid\!\sim \neg\circ\Box[Q]_v$ and the induction hypothesis for assertion 5 imply $\vdash \Box\neg\Box[Q]_v$, and therefore $\vdash \neg\Box[Q]_v$, by (ax1). If $\Box[Q]_v$ is from $\mathcal{F}$, this is exactly the assertion. If $\Box[Q]_v$ is from

$\mathcal{F}$, we also have $\mid\!\sim \Box[Q]_v$, and therefore $\vdash \Box[Q]_v$, by the induction hypothesis for assertion 1. Together, we obtain $\vdash$ **false** as desired.

(pax4.4) : Then $\mathcal{A} = \mathcal{P} = \langle\langle v, \neg\circ v\rangle \to Q, \langle\neg v, \circ v\rangle \to Q, \circ\Box[Q]_v\rangle$ and P is $\Box[Q]_v$, for some pre-formula Q. The assumptions imply, in particular, that $\mid\!\sim \circ\Box[Q]_v$ and $\mid\!\sim \neg\Box[Q]_v$. Using the induction hypotheses for assertions 4 and 1, we obtain $\vdash \Box\Box[Q]_v$ and $\vdash \neg\Box[Q]_v$, hence $\vdash$ **false** follows by (ax1) and propositional logic.

(pax5) : Then $\mathcal{A} = \mathcal{P} = \langle\circ\Box G\rangle$ and P is $\Box[\circ G]_v$, for some formula G. We have $\mid\!\sim \circ\Box G$, and therefore $\vdash \Box G$, using the induction hypothesis for assertion 4 and (ax1). By (T6), we obtain $\vdash \Box[\circ G]_v$. On the other hand, from the assumption $\mid\!\sim \neg\Box[\circ G]_v$ and the induction hypothesis for assertion 1 we obtain $\vdash \neg\Box[\circ G]_v$, hence $\vdash$ **false**.

(pmp) : Then we have $\mid\!\sim P_0$, $\mid\!\sim \langle P_0, A_1, \ldots, A_{n+m}\rangle \to P$ for some pre-formula P_0 besides the assumptions $\mid\!\sim \neg P$, and $\mid\!\sim \mathcal{F}^i \to P_i$ for $i = 1, \ldots, m$. We may therefore apply the induction hypothesis for assertion 2 to obtain the assertion.

(pre) : In this case we have $\vdash \mathcal{A} \to P$ just before the application of (pre). In particular, P and all P_i must be formulas, so we may replace "$\mid\!\sim$" by "$\vdash$" in the assumptions $\mid\!\sim \neg P$ and $\mid\!\sim \mathcal{F}^i \to P_i$, using the induction hypothesis for assertion 1. The assertion, $\vdash \neg(F_1 \wedge \ldots \wedge F_n)$ follows by propositional logic.

(nex) : Then $\mathcal{A}$ must be $\langle\rangle$, and P is $\circ G$, for some formula G such that $\vdash G$. On the other hand, we have $\mid\!\sim \neg\circ G$, and therefore $\vdash \Box\neg G$, by the induction hypothesis for assertion 5. The assertion, $\vdash$ **false**, follows via (ax1) and propositional logic.

3. The proof parallels that of assertion 2, except that axioms (pax1.2), (pax1.4), (pax2), (pax3), (pax4.3), (pax4.4), (pax5), and rule (nex) do not apply.

4. Consider the last step in the assumed derivation of $\mid\!\sim \mathcal{A} \to \circ F$.

(pax0.1) : Then $\mathcal{A} = \langle\circ F, A\rangle$ for some A, and clearly, $\circ F$ is from $\mathcal{P}$. Therefore we have $\mid\!\sim \circ F$, and the induction hypothesis for assertion 4 yields $\vdash \Box F$. The assertion, $\vdash \mathcal{F} \to \Box F$, follows propositionally.

(pax0.2) : Then $\mathcal{A} = \langle\langle A, B\rangle \to \circ F, A \to B, A\rangle$. Clearly, $\langle A, B\rangle \to \circ F$ is from $\mathcal{P}$, so we have $\mid\!\sim \langle A, B\rangle \to \circ F$.

(a) If $A \to B$ and A are from $\mathcal{F}$, then A and B are formulas, and from the induction hypothesis for assertion 4 (for $\mathcal{F}' = \langle A, B\rangle$ and $\mathcal{P}' = \langle\rangle$) we obtain $\vdash \langle A, B\rangle \to \Box F$. The assertion, $\vdash \langle A \to B, A\rangle \to \Box F$ follows by propositional reasoning.

(b) If $A \to B$ is from $\mathcal{F}$ and A is from $\mathcal{P}$, then again A and B are formulas, so we may again conclude $\vdash \langle A, B\rangle \to \Box F$. Moreover, we also have $\mid\!\sim$ $(A \to B) \to A$, from which we obtain $\vdash (A \to B) \to A$, by the induction hypothesis for assertion 1. Propositional reasoning shows $\vdash A$, hence also $\vdash (A \to B) \to (A \wedge B)$. With $\vdash \langle A, B \rangle \to \Box F$, this gives $\vdash (A \to B) \to \Box F$, as desired.

(c) If $A \to B$ is from $\mathcal{P}$ and A is from $\mathcal{F}$, we have $\mid\!\sim A \to B$, as well as $\mid\!\sim \langle A, B \rangle \to \circ F$. Applying the induction hypothesis for assertion 4 (choosing $\mathcal{F}' = \langle A \rangle$ and $\mathcal{P}' = \langle B \rangle$) we obtain $\vdash A \to \Box F$, as desired.

(d) If $A \to B$ and A are both from $\mathcal{P}$, we have $\mid\!\sim A \to B$ and $\vdash A$, so by applying (pmp) we obtain a derivation of the simpler formulas $\mid\!\sim A$, $\mid\!\sim B$, and $\mid\!\sim \langle A, B \rangle \to \circ F$ whose overall length is the same as that of the given derivation, so we may apply the induction hypothesis for assertion 4 to conclude $\vdash \Box F$, as desired.

(pax0.3) : Then $\mathcal{A} = \langle \neg\circ F \to \neg A, A \rangle$, for some A. Clearly, $\neg\circ F \to \neg A$ must be from $\mathcal{P}$, so we have $\mid\!\sim \neg\circ F \to \neg A$.

(a) If A is from $\mathcal{F}$ then $\neg A$ is a formula. By the induction hypothesis for assertion 7, it follows that $\vdash \neg\neg A \to \Box F$, hence also $\vdash A \to \Box F$, by propositional reasoning.

(b) If A is from $\mathcal{P}$ then we have $\mid\!\sim A$, and the induction hypothesis for assertion 15 gives $\vdash F$, and therefore also $\vdash \Box F$, using rule (alw).

(pax0.4) : Then $\mathcal{A} = \langle \neg\circ F \to A, \neg A \rangle$, for some A, and $\neg\circ F \to A$ must be from $\mathcal{P}$, so we have $\mid\!\sim \neg\circ F \to A$.

(a) If $\neg A$ is from $\mathcal{F}$ then A is a formula. By the induction hypothesis for assertion 7, it follows that $\vdash \neg A \to \Box F$, as desired.

(b) If $\neg A$ is from $\mathcal{P}$ then we have $\mid\!\sim \neg A$, and the induction hypothesis for assertion 14 gives $\vdash F$, and therefore also $\vdash \Box F$, using rule (alw).

(pax0.5), (pax0.6) : not applicable.

(pax1.1) : not applicable.

(pax1.2) : In this case, $\mathcal{A} = \mathcal{P} = \langle \neg\circ G \rangle$, and F is $\neg G$, for some formula G. By assumption, we have $\mid\!\sim \neg\circ G$, and therefore $\vdash \Box\neg G$, by the induction hypothesis for assertion 5, which is exactly what we want to show.

(pax1.3) : not applicable

(pax1.4) : Then $\mathcal{A} = \mathcal{P} = \langle \neg\circ\neg F \rangle$. From $\mid\!\sim \neg\circ\neg F$ and the induction hypothesis for assertion 5 we conclude $\vdash \Box\neg\neg F$, and therefore also $\vdash \Box F$ as desired.

(pax2) : Then $\mathcal{A} = \mathcal{P} = \langle \circ(G \to F), \circ G \rangle$, for some formula G, and the assumptions are $\mid\!\sim \circ(G \to F)$ and $\mid\!\sim \circ G$. By the induction hypothesis for assertion 4 we obtain $\vdash \Box(G \to F)$ and $\vdash \Box G$. Using (ax1), we get $\vdash G \to F$ and $\vdash G$, so (mp) gives $\vdash F$, and the assertion, $\vdash \Box F$ follows by rule (alw).

(pax3) : Then $\mathcal{A} = \langle \Box G \rangle$ and F is $\Box G$, for some formula G. If $\Box G$ is in $\mathcal{F}$, then the assertion is $\vdash \Box G \to \Box\Box G$, which follows from (T1). If $\Box G$ is in $\mathcal{P}$, we have $\mid\!\sim \Box G$, and therefore $\vdash \Box G$, by the induction hypothesis for assertion 1. The assertion, $\vdash \Box\Box G$, follows by rule (alw).

(pax4.1) : Then $\mathcal{A} = \langle \Box[\circ F]_v, v, \neg\circ v \rangle$ where, clearly, $\neg\circ v$ is from $\mathcal{P}$.

(a) If $\Box[\circ F]_v$ and v are both from $\mathcal{F}$, we have $\mid\!\sim \langle \Box[\circ F]_v, v \rangle \rightarrow \neg\circ v$, and the induction hypothesis for assertion 5 implies $\vdash \langle \Box[\circ F]_v, v \rangle \rightarrow \Box\neg v$. The assertion, $\vdash \langle \Box[\circ F]_v, v \rangle \rightarrow \Box F$, follows with the help of (ax1) and propositional logic.

(b) If $\Box[\circ F]_v$ is from $\mathcal{F}$ and v is from $\mathcal{P}$, we have both $\mid\!\sim \Box[\circ F]_v \rightarrow v$ and $\mid\!\sim \Box[\circ F]_v \rightarrow \neg\circ v$. Using the induction hypotheses for assertions 1 and 4, we conclude $\vdash \Box[\circ F]_v \rightarrow v$ and $\vdash \Box[\circ F]_v \rightarrow \Box\neg v$. The assertion $\vdash \Box[\circ F]_v \rightarrow \Box F$ now follows by axiom (ax1) and propositional logic.

(c) If $\Box[\circ F]_v$ is from $\mathcal{P}$ and v is from $\mathcal{F}$, we have $\mid\!\sim v \rightarrow \neg\circ v$, and therefore $\vdash v \rightarrow \Box\neg v$, by the induction hypothesis for assertion 5. The assertion $\vdash v \rightarrow \Box F$ again follows by (ax1) and propositional logic.

(d) If $\Box[\circ F]_v$ and v are both from $\mathcal{P}$, we have in particular $\mid\!\sim v$ and $\mid\!\sim \neg\circ v$. Using the induction hypotheses for assertions 1 and 5, we conclude $\vdash v$ and $\vdash \Box\neg v$, hence $\vdash \Box F$ follows with the help of (ax1) and propositional logic.

(pax4.2) : similar.

(pax4.3) : Then $\mathcal{A} = \langle \Box[P]_v \rangle$ and F is $\Box[P]_v$, for some P. If $\Box[P]_v$ is from $\mathcal{F}$, we have to show $\vdash \Box[P]_v \rightarrow \Box\Box[P]_v$, which follows from (T2). On the other hand, if $\Box[P]_v$ is from $\mathcal{P}$, we have $\mid\!\sim \Box[P]_v$, and therefore $\vdash \Box[P]_v$, by the induction hypothesis for assertion 1. The assertion, $\vdash \Box\Box[P]_v$, again follows via (T2).

(pax4.4), (pax5) : not applicable.

(pmp) : Then we have $\mid\!\sim P_0$ and $\mid\!\sim \langle P_0, A_1, \ldots, A_{m+n} \rangle \rightarrow \circ F$, for some preformula P_0, and the assertion follows by the induction hypothesis for assertion 4.

(pre) : not applicable.

(nex) : Then $\mathcal{A} = \langle\rangle$, and we have $\vdash F$ prior to the application of (nex). The assertion, $\vdash \Box F$, follows by rule (alw).

5. The proof is very similar to that of assertion 4.

6. Consider the last step in the derivation of $\mid\!\sim \mathcal{B}^k \rightarrow G$.

(pax0.1) : The axiom is $\mid\!\sim \langle G, \circ F \rangle \rightarrow G$, and by assumption we have $\mid\!\sim G$, where G is a formula. The induction hypothesis for assertion 1 gives $\vdash G$, and the assertion, $\vdash \neg G \rightarrow \Box\neg F$, follows by propositional logic.

(pax0.2) : The axiom instance must be $\mid\!\sim \langle \langle \circ F, A \rangle \rightarrow G, \circ F \rightarrow A, \circ F \rangle \rightarrow G$, for some A, and by assumption we have $\mid\!\sim \langle \circ F, A \rangle \rightarrow G$ and $\circ F \rightarrow A$. So we may invoke the induction hypothesis for assertion 6, choosing $\mathcal{P}' = \langle A \rangle$ to deduce $\vdash \neg G \rightarrow \Box\neg F$, as desired.

(pax0.3) : The axiom instance must be $\mid\!\sim \langle\neg G \rightarrow \neg\circ F, \circ F\rangle \rightarrow G$, and by assumption we have $\mid\!\sim \neg G \rightarrow \neg\circ F$. By the induction hypothesis for assertion 5, we obtain $\vdash \neg G \rightarrow \Box\neg F$, as desired.

(pax0.4) : not applicable.

(pax0.5) : The axiom instance must be $\mid\!\sim \langle H \rightarrow \neg\circ F, \circ F\rangle \rightarrow \neg H$, where G is $\neg H$. By assumption, we have $\mid\!\sim H \rightarrow \neg\circ F$, and the induction hypothesis for assertion 5 gives $\vdash H \rightarrow \Box\neg F$. The assertion, $\vdash \neg\neg H \rightarrow \Box\neg F$, follows by propositional reasoning.

(pax0.6) : not applicable.

(pax1.1)–(pax1.4), (pax2), (pax3) : Neither of these axioms applies, because they are not of the shape $\mathcal{A} \rightarrow G$ where G is a formula.

(pax4.1) : not applicable, because it does not contain a pre-formula $\circ F$ to the left of $\rightarrow$.

(pax4.2) : The axiom is $\mid\!\sim \langle\Box[G]_v, \neg v, \circ v\rangle \rightarrow G$, where F is v. By assumption, we have $\mid\!\sim \neg v$, so we may invoke the induction hypothesis for assertion 1 to conclude $\vdash \neg v$. Using rule (alw), we obtain $\vdash \Box\neg v$, and therefore also $\vdash \neg G \rightarrow \Box\neg v$, using propositional logic.

(pax4.3) : not applicable.

(pax4.4) : The instance is $\langle\langle v, \neg\circ v\rangle \rightarrow P, \langle\neg v, \circ v\rangle \rightarrow P, \circ\Box[P]_v\rangle \rightarrow \Box[P]_v$, where both F and G are $\Box[P]_v$, so we need to show $\vdash \neg\Box[P]_v \rightarrow \Box\neg\Box[P]_v$. By assumption, we have $\mid\!\sim \langle v, \neg\circ v\rangle \rightarrow P$ and $\mid\!\sim \langle\neg v, \circ v\rangle \rightarrow P$, hence also $\mid\!\sim [P]_v$, by propositional logic. Using rule (sq), we obtain $\vdash \Box[[P]_v]_v$, which implies $\vdash \Box[P]_v$, using (T3). The assertion follows propositionally.

(pax5) : The axiom is $\mid\!\sim \circ\Box H \rightarrow \Box[\circ H]_v$, for some formula H, so F is $\Box H$ and G is $\Box[\circ H]_v$. The assertion, $\vdash \neg\Box[\circ H]_v \rightarrow \neg\Box H$, is just the contrapositive of (T6).

(pmp) : Then we already have $\mid\!\sim \langle P_0, P_1, \ldots, P_k, \circ F, P_{k+1}, \ldots, P_m\rangle \rightarrow G$ and $\mid\!\sim P_0$ prior to the application of (pmp), and the induction hypothesis for assertion 6 gives $\vdash \neg G \rightarrow \Box\neg F$ as desired.

(pre), (nex) : not applicable.

7. The proof is very similar to that of assertion 6.

8. Consider the last step in the derivation of $\mid\!\sim \mathcal{B}^k \rightarrow \circ G$.

(pax0.1) : The axiom instance is $\mid\!\sim \langle\circ G, A\rangle \rightarrow \circ G$, for some A.

(a) If F is G (that is, $k = 0$), then the assertion is $\vdash G \rightarrow G$, which holds by (ax0).

(b) If $\circ G$ is from $\mathcal{P}$ and A is $\circ F$ (that is, $k = 1$), then we have $\mid\!\sim \circ G$ by assumption. The induction hypothesis for assertion 4 yields $\vdash \Box G$, and the assertion, $\vdash F \rightarrow G$, follows by axiom (ax1) and propositional logic.

(pax0.2) : The axiom instance must be $\langle\langle\circ F, A\rangle \rightarrow \circ G, \circ F \rightarrow A, \circ F\rangle \rightarrow \circ G$, where $\langle\circ F, A\rangle \rightarrow \circ G$ and $\circ F \rightarrow A$ are from $\mathcal{P}$. We have the assumptions $\mid\!\sim \langle\circ F, A\rangle \rightarrow \circ G$ and $\mid\!\sim \circ F \rightarrow A$, and may apply the induction hypothesis for assertion 8 (choosing $\mathcal{P}' = \langle A\rangle$) to conclude $\vdash F \rightarrow G$ as desired.

(pax0.3) : The axiom instance is $\mid\!\sim \langle\neg\circ G \rightarrow \neg\circ F, \circ F\rangle \rightarrow \circ G$. By assumption we have $\mid\!\sim \neg\circ G \rightarrow \neg\circ F$, and the induction hypothesis for assertion 11 yields $\vdash \neg G \rightarrow \neg F$, hence $\vdash F \rightarrow G$ follows propositionally.

(pax0.4)–(pax0.6) : not applicable.

(pax1.1)–(pax1.4) : not applicable, because neither of these has a pre-formula of the shape $\circ H$ to the left and to the right of $\rightarrow$.

(pax2) : The axiom instance is $\mid\!\sim \langle\circ(H \rightarrow G), \circ H\rangle \rightarrow \circ G$, for some formula H.

(a) If F is $H \rightarrow G$ and $\circ H$ is from $\mathcal{P}$ (that is, $k = 0$), we need to show $\vdash (H \rightarrow G) \rightarrow G$. By assumption, we have $\mid\!\sim \circ(H \rightarrow G) \rightarrow \circ H$, and the induction hypothesis for assertion 8 gives $\vdash (H \rightarrow G) \rightarrow H$, and therefore $\vdash H$, by propositional logic. The assertion follows again by propositional reasoning.

(b) If $\circ(H \rightarrow G)$ is from $\mathcal{P}$ and F is H (that is, $k = 1$) then by assumption we have $\mid\!\sim \circ(H \rightarrow G)$, hence the induction hypothesis for assertion 4 and (ax1) give $\vdash H \rightarrow G$ as desired.

(pax3), (pax4.1) : not applicable, because there is no formula $\circ F$ to the left of $\rightarrow$.

(pax4.2) : The instance is $\mid\!\sim \langle\Box[\circ G]_v, \neg v, \circ v\rangle \rightarrow \circ G$, where F is v. By assumption, we have $\mid\!\sim \neg v$, and therefore $\vdash \neg v$, by the induction hypothesis for assertion 1. Hence, $\vdash v \rightarrow G$ follows propositionally.

(pax4.3), (pax4.4), (pax5) : not applicable.

(pmp) : by the usual argument.

(pre), (nex) : not applicable.

9. The argument is similar to the preceding assertion, except for a few cases. Again, consider the last step in the derivation of $\mid\!\sim \mathcal{B}^k \rightarrow \neg\circ G$.

(pax0.1) : The axiom is $\mid\!\sim \langle\neg\circ G, \circ F\rangle \rightarrow \neg\circ G$. Since we know $\mid\!\sim \neg\circ G$ by assumption, the induction hypothesis for assertion 5 yields $\vdash \Box\neg G$. The assertion, $\vdash F \rightarrow \neg G$, follows by (ax1) and propositional logic.

(pax0.2) : The axiom must be $\mid\!\sim \langle\langle\circ F, A\rangle \rightarrow \neg\circ G, \circ F \rightarrow A, \circ F\rangle \rightarrow \neg\circ G$, for some A. By assumption we have $\mid\!\sim \langle\circ F, A\rangle \rightarrow \neg\circ G$ and $\mid\!\sim \circ F \rightarrow A$. Applying the induction hypothesis for assertion 9, we conclude $\vdash F \rightarrow \neg G$, as desired.

(pax0.3) : The axiom is $\mid\!\sim \langle\neg\neg\circ G \rightarrow \neg\circ F, \circ F\rangle \rightarrow \neg\circ G$, and by assumption we have $\mid\!\sim \neg\neg\circ G \rightarrow \neg\circ F$. By lemma A.1 we can show $\mid\!\sim \circ G \rightarrow \neg\circ F$ by a derivation whose length does not exceed that of the assumed derivation of

$\mathrel{|\!\!\sim} \neg\neg\circ G \rightarrow \neg\circ F$, so we may use the induction hypothesis for assertion 9 to conclude $\vdash G \rightarrow \neg F$. The assertion, $\vdash F \rightarrow \neg G$, follows by propositional logic.

(pax0.4) : The axiom is $\mathrel{|\!\!\sim} \langle \circ G \rightarrow \neg\circ F, \circ F\rangle \rightarrow \neg\circ G$, and by assumption we have $\mathrel{|\!\!\sim} \circ G \rightarrow \neg\circ F$, hence the assertion follows as in the preceding case.

(pax0.5), (pax0.6) : not applicable.

(pax1.1) : The axiom is $\mathrel{|\!\!\sim} \circ\neg G \rightarrow \neg\circ G$, and we have to show $\vdash \neg G \rightarrow \neg G$, which is an instance of (ax0).

(pax1.2), (pax1.3) : not applicable.

(pax1.4) : The axiom is $\mathrel{|\!\!\sim} \circ F \rightarrow \neg\circ\neg F$, where G is $\neg F$, and we need to show $\vdash F \rightarrow \neg\neg F$, which holds by (ax0).

(pax2), (pax3) : not applicable, because the pre-formula to the right of $\rightarrow$ is not of the shape $\neg\circ G$.

(pax4.1) : not applicable, because there is no pre-formula $\circ F$ to the left of $\rightarrow$.

(pax4.2) : The axiom is $\mathrel{|\!\!\sim} \langle \Box[\neg\circ G]_v, \neg v, \circ v\rangle \rightarrow \neg\circ G$, where F is v. By assumption we have $\mathrel{|\!\!\sim} \neg v$, which implies $\vdash \neg v$, by the induction hypothesis for assertion 1. Hence, $\vdash v \rightarrow \neg G$ follows by propositional logic.

(pax4.3), (pax4.4), (pax5) : not applicable.

(pmp) : by the usual argument.

(pre), (nex) : not applicable.

10. The proof is similar as for assertions 8 and 9.

11. The proof is similar as for assertions 8 and 9.

12. Consider the last step in the derivation of $\mathrel{|\!\!\sim} \mathcal{B}^k \rightarrow P$.

(pax0.1) : The axiom is $\mathrel{|\!\!\sim} \langle P, A\rangle \rightarrow P$.

(a) If P is $\circ F$ (that is, $k = 0$) then by assumption we have $\mathrel{|\!\!\sim} \neg P$, that is, $\mathrel{|\!\!\sim} \neg\circ F$. By the induction hypothesis for assertion 5 it follows that $\vdash \Box\neg F$, and therefore $\vdash \neg F$ by (ax1).

(b) If A is $\circ F$ and P is from $\mathcal{P}$ (that is, $k = 1$) then we have both $\mathrel{|\!\!\sim} P$ and $\mathrel{|\!\!\sim} \neg P$, and the induction hypothesis for assertion 2 implies $\vdash$ **false**. Hence, $\vdash \neg F$ follows by propositional logic.

(pax0.2) : The axiom must be $\mathrel{|\!\!\sim} \langle\langle \circ F, A\rangle \rightarrow P, \circ F \rightarrow A, \circ F\rangle \rightarrow P$, where both $\langle \circ F, A\rangle \rightarrow P$ and $\circ F \rightarrow A$ are from $\mathcal{P}$. So we have $\mathrel{|\!\!\sim} \langle \circ F, A\rangle \rightarrow P$ as well as $\mathrel{|\!\!\sim} \circ F \rightarrow A$ and $\mathrel{|\!\!\sim} \neg P$, and the induction hypothesis for assertion 12 implies $\vdash \neg F$, as desired.

(pax0.3) : The axiom is $\mathrel{|\!\!\sim} \langle \neg P \rightarrow \neg\circ F, \circ F\rangle \rightarrow P$. By assumption, we have $\mathrel{|\!\!\sim} \neg P \rightarrow \neg\circ F$ and $\mathrel{|\!\!\sim} \neg P$, so we may apply the induction hypothesis for assertion 5 to conclude $\vdash \Box\neg F$, and therefore $\vdash \neg F$, by (ax1).

(pax0.4) : The axiom is $\mid\!\sim \langle Q \to \neg \circ F, \circ F \rangle \to \neg Q$, where P is $\neg Q$. By asumption, we have both $\mid\!\sim Q \to \neg \circ F$ and $\mid\!\sim \neg\neg Q$. Lemma A.1 ensures that there is a derivation of $\mid\!\sim Q$ whose length does not exceed that of $\mid\!\sim \neg\neg Q$. We may therefore apply the induction hypothesis for assertion 5 to conclude $\vdash \Box\neg F$, and $\vdash \neg F$ follows by (ax1).

(pax0.5), (pax0.6) : not applicable.

(pax1.1) : The axiom is $\mid\!\sim \circ\neg G \to \neg \circ G$, where P is $\neg \circ G$ and F is $\neg G$. By assumption, we have $\mid\!\sim \neg\neg \circ G$, and lemma A.1 ensures that there is a derivation of $\mid\!\sim \circ G$ whose length is at most that of the derivation of $\mid\!\sim \neg\neg \circ G$. Now, the induction hypothesis for assertion 4 implies $\vdash \Box G$, and $\vdash \neg\neg G$ follows by (ax1) and propositional reasoning.

(pax1.2), (pax1.3) : not applicable.

(pax1.4) : The axiom is $\mid\!\sim \circ F \to \neg \circ \neg F$, where P is $\neg \circ \neg F$. By assumption, we have $\mid\!\sim \neg\neg \circ \neg F$. By lemma A.1 and the induction hypothesis for assertion 4 we conclude $\vdash \Box\neg F$, hence $\vdash \neg F$ follows by (ax1).

(pax2) : The axiom is $\mid\!\sim \langle \circ(G \to H), \circ G \rangle \to \circ H$, where P is $\circ H$. By assumption, we have $\mid\!\sim \neg \circ H$, and therefore the induction hypothesis for assertion 5 and (ax1) imply $\vdash \neg H$.

(a) If F is $G \to H$ and $\circ G$ is from $\mathcal{P}$ (that is, $k = 0$) then we have the assumption $\mid\!\sim \circ(G \to H) \to \circ G$. By the induction hypothesis for assertion 8, we obtain $\vdash (G \to H) \to G$, and therefore $\vdash G$, by propositional logic. Together with $\vdash \neg H$, this shows $\vdash \neg(G \to H)$, as desired.

(b) If $\circ(G \to H)$ is from $\mathcal{P}$ and F is G (that is, $k = 1$), we use the assumption $\mid\!\sim \circ(G \to H)$ and the induction hypothesis for assertion 4, together with (ax1), to obtain $\vdash G \to H$. Combined with $\vdash \neg H$, this shows $\vdash \neg G$.

(pax3), (pax4.1) : not applicable.

(pax4.2) : The axiom is $\mid\!\sim \langle \Box[P]_v, \neg v, \circ v \rangle \to P$, where F is v. In particular, we have $\mid\!\sim \neg v$ by assumption, and the desired conclusion $\vdash \neg v$ follows by the induction hypothesis for assertion 1.

(pax4.3) : not applicable.

(pax4.4) : The axiom is $\mid\!\sim \langle \langle v, \neg \circ v \rangle \to Q, \langle \neg v, \circ v \rangle \to Q, \circ\Box[Q]_v \rangle \to \Box[Q]_v$, where F and P are both $\Box[Q]_v$. In particular, we have the assumption $\mid\!\sim \neg P$, that is, $\mid\!\sim \neg\Box[Q]_v$, which by the induction hypothesis for assertion 1 implies $\vdash \neg\Box[Q]_v$, as desired.

(pax5) : The axiom is $\mid\!\sim \circ\Box G \to \Box[\circ G]_v$, where F is $\Box G$ and P is $\Box[\circ G]_v$. By assumption, we have $\mid\!\sim \neg\Box[\circ G]_v$, and therefore $\vdash \neg\Box[\circ G]_v$, using the induction hypothesis for assertion 1. The assertion, $\vdash \neg\Box G$, follows by (T6).

(pmp) : by the usual argument.

(pre), (nex) : not applicable.

13. Again, consider the last step in the derivation of $\mid\!\sim \mathcal{B}^k \to \neg P$.

(pax0.1) : The axiom is $\mid\!\sim \langle \neg P, A\rangle \to \neg P$, and clearly, $\neg P$ must be from $\mathcal{P}$. So we have $\mid\!\sim \neg P$ and $\mid\!\sim P$, hence the induction hypothesis for assertion 2 implies $\vdash$ **false**. The assertion, $\vdash \neg F$, follows by propositional logic.

(pax0.2) : The axiom must be $\mid\!\sim \langle\langle \circ F, A\rangle \to \neg P, \circ F \to A, \circ F\rangle \to \neg P$, where both $\langle \circ F, A\rangle \to \neg P$ and $\circ F \to A$ are from $\mathcal{P}$. Therefore $\mid\!\sim \langle \circ F, A\rangle \to \neg P$ as well as $\mid\!\sim \circ F \to A$ and $\mid\!\sim P$ all hold, and the induction hypothesis for assertion 13 implies $\vdash \neg F$, as desired.

(pax0.3) : The axiom is $\mid\!\sim \langle \neg\neg P \to \neg\circ F, \circ F\rangle \to \neg P$. By assumption, we have $\mid\!\sim \neg\neg P \to \neg\circ F$ and $\mid\!\sim P$. Using lemma A.1 and the induction hypothesis for assertion 5, we may conclude $\vdash \Box\neg F$, and therefore $\vdash \neg F$, by (ax1).

(pax0.4) : The axiom is $\mid\!\sim \langle P \to \neg\circ F, \circ F\rangle \to \neg P$. By asumption, we have both $\mid\!\sim P \to \neg\circ F$ and $\mid\!\sim P$, and conclude $\vdash \neg F$ by the induction hypothesis for assertion 5 and (ax1).

(pax0.5), (pax0.6) : not applicable.

(pax1.1) : The axiom is $\mid\!\sim \circ\neg G \to \neg\circ G$, where P is $\circ G$ and F is $\neg G$. By assumption, we have $\mid\!\sim \circ G$, and therefore $\vdash G$, by the induction hypothesis for assertion 4 and (ax1). The assertion, $\vdash \neg\neg G$, follows by propositional logic.

(pax1.2), (pax1.3) : not applicable.

(pax1.4) : The axiom is $\mid\!\sim \circ F \to \neg\circ\neg F$, where P is $\circ\neg F$. By assumption, we have $\mid\!\sim \circ\neg F$, and therefore $\vdash \neg F$, using the induction hypothesis for assertion 4 and (ax1).

(pax2), (pax3), (pax4.1) : not applicable.

(pax4.2) : The axiom is $\mid\!\sim \langle \Box[\neg P]_v, \neg v, \circ v\rangle \to \neg P$, where F is v. In particular, we have $\mid\!\sim \neg v$ by assumption, and the desired conclusion $\vdash \neg v$ follows by the induction hypothesis for assertion 1.

(pax4.3), (pax4.4), (pax5) : not applicable.

(pmp) : by the usual argument.

(pre), (nex) : not applicable.

14. The proof is similar as for assertions 12 and 13.

15. The proof is similar as for assertions 12 and 13. Q.E.D.

Theorem 2.8. *For any set $\mathcal{F}$ of formulas and any formula F:*

$$\mathcal{F} \vdash F \quad \textit{iff} \quad \mathcal{F} \mathrel{|\!\sim} F \quad \textit{iff} \quad \mathcal{F} \mathrel{|\!\sim} \circ F$$

Proof. For both assertions, the "only if" direction is immediate by rules (pre) and (nex). For the other direction, assume that $\mathcal{F} \mathrel{|\!\sim} F$. This derivation can make use of only finitely many formulas, say $F_1, \ldots, F_n$, from $\mathcal{F}$. By theorem 2.7, there is also a derivation of $\mathrel{|\!\sim} \langle \Box F_1, \ldots, \Box F_n \rangle \to F$. Assertion 1 of lemma A.2 implies $\vdash \langle \Box F_1, \ldots, \Box F_n \rangle \to F$, and $\mathcal{F} \vdash F$ follows again by theorem 2.7. A similar argument, based on assertion 4 of lemma A.2, proves the second assertion. Q.E.D.

A.3 Derivations in $\Sigma^h_{\mathrm{TLA}^*}$

The following derivations give an impression of the use of $\Sigma^h_{\mathrm{TLA}^*}$.

(H1) $\Box[A]_v, \Box[A \to B]_v \overset{h}{\vdash} \Box[B]_v$

(1)	$\Box[A]_v$	(assumption)
(2)	$\Box[A \to B]_v$	(assumption)
(3)	$\Box[A \to B]_v \to (\Box[A]_v \to \Box[B]_v)$	(hx5)
(4)	$\Box[A]_v \to \Box[B]_v$	(hmp)(2)(3)
(5)	$\Box[B]_v$	(hmp)(1)(4)

(H2) $F \overset{h}{\vdash} \Box[F]_v$

(1)	F	(assumption)
(2)	$\Box F$	(alw)(1)
(3)	$\Box F \to \Box[F]_v$	(hx2)
(4)	$\Box[F]_v$	(hmp)(2)(3)

(H3) $\Box[A \to B]_v, \Box[B \to C]_v \overset{h}{\vdash} \Box[A \to C]_v$

(1)	$\Box[A \to B]_v$	(assumption)
(2)	$\Box[B \to C]_v$	(assumption)
(3)	$\Box[(A \to B) \to ((B \to C) \to (A \to C))]_v$	(hx7)
(4)	$\Box[(B \to C) \to (A \to C)]_v$	(H1)(1)(3)
(5)	$\Box[A \to C]_v$	(H1)(2)(4)

(H4) $\Box[[A]_v \to A]_v$

(1)	$\Box[v' \neq v \to ([A]_v \to A)]_v$	(hx7)
(2)	$\Box[v' \neq v]_v$	(hx6)
(3)	$\Box[[A]_v \to A]_v$	(H1)(1)(2)

(H5) $\Box[\Box F \to \bigcirc\Box F]_v$

(1)	$\Box F \to \Box[\bigcirc\Box F]_v$	(hx3)
(2)	$\Box[\Box F \to \Box[\bigcirc\Box F]_v]_v$	(H2)(1)
(3)	$\Box[\Box[\bigcirc\Box F]_v \to [\bigcirc\Box F]_v]_v$	(hx10)
(4)	$\Box[\Box F \to [\bigcirc\Box F]_v]_v$	(H3)(2)(3)
(5)	$\Box[[\bigcirc\Box F]_v \to \bigcirc\Box F]_v$	(H4)
(6)	$\Box[\Box F \to \bigcirc\Box F]_v$	(H3)(4)(5)