Progress Measures and Finite Arguments for Infinite Computations

Nils Klarlund
Ph.D Thesis

90-1153
September 1990

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

PROGRESS MEASURES AND FINITE ARGUMENTS FOR INFINITE COMPUTATIONS

A Dissertation
Presented to the Faculty of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by
Nils Klarlund
August 1990

© Nils Klarlund 1990
ALL RIGHTS RESERVED

PROGRESS MEASURES AND FINITE ARGUMENTS FOR INFINITE COMPUTATIONS

Nils Klarlund, Ph.D.
Cornell University 1990

We establish principles for proving properties about infinite computations by reasoning about finite ones. We apply these principles to show that for a wide variety of verification problems—involving nondeterminism, fairness, and liveness—there are assertional verification methods that directly relate program and specification.

Most previous research relies on transformations of programs in order to reduce a verification problem to problems that can be solved using classical techniques such as refinement mappings and well-founded orderings. *Progress measures*, the key innovation of this thesis, provide direct, syntax-independent verification techniques for a wide range of specifications.

We exhibit progress measures for the language containment problem for nondeterministic automata; for verification with general fairness constraints; and for verification of very general specifications, including infinitary temporal logics. We obtain an optimal solution (in a recursion-theoretic sense) to the problem of verifying infinite computations by reasoning about finite ones. This result establishes a link between descriptive set theory and the theory of verification.

Biographical Sketch

Nils Klarlund was born 1962 in Copenhagen, Denmark and graduated in 1986 from Aarhus University, Denmark with a Cand. Scient. degree after studies at Université des Sciences et Techniques du Languedoc, Montpellier, France; Technical University of Denmark; and University of Copenhagen. He has been at Cornell University since 1986.

To my parents

Acknowledgements

Professor Dexter Kozen's insight and encouragement have been indispensable for turning my work in the right directions. His acute observations have greatly sharpened and simplified the results in this thesis. It is a privilege to have an advisor whose enthusiasm for so many aspects of a large field cannot but inspire one to keep going.

I am also grateful for having had the opportunity to work with Professor Fred Schneider, who served on my committee and introduced me to the problems that motivated this thesis. Also, thanks to Anil Nerode and Richard Shore for serving on my committee.

Other researchers have been very helpful in discussing work on verification; special thanks to Martín Abadi, Bowen Alpern, Nissim Francez, Alexander Kechris, Leslie Lamport, Michael Main, Wiktor Marek, and Aravinda Prasad Sistla.

Thanks to all fellow students who through their friendship or assistance have helped make four enjoyable years at Cornell, to mention a few: David Basin, Richard Chapman, Charles Elkan, Bruce Esrig, Mike Manyan, Steve Mitchell, Steve Rapkin, Michael Schwartzbach, Mike Slifker, and George Torres. Thanks to Liên Tran for making the last year at Cornell especially pleasant.

From my years at Danish universities, Professors Dines Bjørner, Erik Meineche Schmidt, Carsten Thomassen, and Peter Øhrstrøm spurred me through their teaching and guidance to probe the mathematics of computation.

Finally I most gratefully acknowledge the financial support of Aarhus University, the Thanks to Scandinavia Foundation, Inc, and Forskerakademiet, Aarhus.

Table of Contents

1 Introduction 1
- 1.1 Thesis Overview . . . 5
- 1.2 Notation . . . 6

I Issues 7

2 Reasoning with Nondeterminism 8
- 2.1 Grammars for Popular Songs . . . 8
- 2.2 Relationships between System and Description . . . 9
- 2.3 Incompleteness of Simple Measures . . . 10

3 Reasoning with Liveness 14
- 3.1 A Persistent Need For Pop Songs . . . 14
- 3.2 The Pop Song Example—Networked Version . . . 17
- 3.3 General Liveness Properties . . . 19

II Solutions 22

4 Safety automata 23
- 4.1 Definitions . . . 24
- 4.2 Safety Properties . . . 25
- 4.3 Measures for Nondeterministic Automata . . . 26
  - 4.3.1 Refinement Measure . . . 27
  - 4.3.2 Prophecy Relation and Measure . . . 28
  - 4.3.3 History Relation and Measure . . . 30
  - 4.3.4 ND Relation and Measure . . . 31
- 4.4 Derivation of Previous Methods . . . 33
  - 4.4.1 Adding History Information . . . 33
  - 4.4.2 Adding Prophecy Information . . . 35
- 4.5 Discussion . . . 36
- 4.6 Historical Notes . . . 38

5 Rabin Conditions 39

5.1 Rabin Conditions 40
5.2 Pointer Trees 40
5.3 Rabin Relation and Measure 42
5.4 Construction of Rabin Measures 44
5.4.1 Colorings of Graphs 44
5.4.2 Proof of Theorem 2 "⇒" 47
5.5 Automata-Theoretic Applications 50
5.5.1 Verification with Deterministic Rabin Automata 50
5.5.2 Verification with Nondeterministic Büchi Automata 51
5.6 Historical Notes 52

6 Infinite Rabin Conditions **54**
6.1 Measures for Infinite Rabin Conditions 54
6.2 Historical Notes 59

7 Liminf Specifications **60**
7.1 Liminf Condition, Relation, and Measure 61
7.2 Another Method for Rabin Conditions 64
7.3 Applications to Descriptive Set Theory 65
7.3.1 Approximation Functions and Limit Operators 66
7.3.2 Liminf Convergence Measure 67
7.3.3 Recursion-Theoretic Discussion 68
7.4 Automata-Theoretic Applications 71
7.5 ¬Liminf Condition and Measure 72
7.6 Historical Notes 73

8 Coanalytic Specifications **75**
8.1 WF Limit Operators 76
8.2 WF Condition, Relation, and Measure 77
8.2.1 WF Convergence Measure 81
8.2.2 Suslin's Theorem 82
8.2.3 Finite Argument Theorem 82
8.3 Borel Automata 84
8.3.1 Proof of the Boundedness Theorem of Section 8.1 85
8.3.2 Borel Sets Are Analytic and Coanalytic 85
8.4 Historical Notes 88

III Examples **89**

9 Termination Under Fairness Constraints **90**
9.1 Examples 90
9.2 A Verification Condition For Fairness 94
9.2.1 Examples (continued) 95
9.3 Connection with Rabin Measures 96

10 Termination under Extreme Fairness Constraints 98
10.1 Example 98
10.2 Verification Condition For Extreme Fairness 99
10.2.1 Example (corrected) 100

11 Conclusion 101

Bibliography 102

List of Figures

1.1 Some Borel classes. 4

3.1 Computation graph G of P. 15
3.2 G with measure. 16
3.3 G containing a livelock situation. 18
3.4 G with livelock and Rabin measure. 20
3.5 A graph with a height assignment. 21

5.1 A pointer tree. 41
5.2 Algorithm `AssignRabin`. 47

6.1 Algorithm `AssignInfRabin`. 55

7.1 Algorithm `AssignLiminf`. 63

8.1 Algorithm `AssignWF`. 79
8.2 Algorithm $\exists$`Merge`. 87

10.1 Program with infinitely many guarded commands. 99

Chapter 1

Introduction

Although computer scientists prefer to reason about finite objects, infinite computations are increasingly studied, for example within program verification, programming languages, fractal theory, and complexity theory. This trend is spurred by computers being mostly used in systems that work continuously, such as text editors, robot control programs, and operating systems. In the limit such systems produce infinite computations.

In this thesis we present principles for proving properties about infinite computations from finite ones. Regard a program P as defining a set, called an *ω-language*, $L(P)$ of infinite computations. For example, if P is a robot control program, $L(P)$ could be the infinite sequences of commands that P may issue. Similarly, a specification S defines a set $L(S)$ of allowed computations. For the robot program above, S could specify only computations for which a certain pattern of commands occurs. To verify that P satisfies S amounts to checking that every computation of P is a computation of S, i.e. that the language containment $L(P) \subseteq L(S)$ holds. This is intrinsically difficult because the question $L(P) \subseteq L(S)$ depends on a universal quantification over infinite objects—a second-order quantification. In terms of recursion theory, the question $L(P) \subseteq L(S)$ is Π^1_2-complete, when P and S are nondeterministic automata, as observed in [47].

Yet to computer scientists, verification methods are often essential to the understanding of programming and semantics. The art of programming is the art of understanding a complex world of events and situations in terms of small steps, each performing an atomic change of state. This is why many methods of verification, such as those based on the assertional techniques of Floyd and Hoare, build

on explaining global behavior from local behavior [8,16,22]. For example, the liveness property that a program terminates can be proved using well-founded sets, as Floyd pointed out in 1967 [16]. A well-founded set $(T, >)$ is a set T with a relation $>$ such that there is no infinite descending chain $t_0 > t_1 > \cdots$. To verify that a program terminates, one assigns an element $\mu(p) \in T$ to each program state p and verifies that whenever there is a transition from p to p', then $\mu(p) > \mu(p')$ holds; Floyd termed such a local requirement a *verification condition*. The assignment $\mu(p)$ is a measure of how close the program is in state p to terminating. Therefore we call μ a *termination measure*.

Use of termination measures—and similar local descriptions, such as *variant* or *bound functions* for proving loop termination—is a cornerstone of program verification [19]. There are a mathematical and a practical reason for this.

The mathematical reason is:

> A program terminates if and only if it has a termination measure.

Thus the method of termination measures is *sound*: if the program has a termination measure, then it terminates (a nonterminating computation $p_0, p_1, \ldots$ would give rise to an infinite descending sequence $\mu(p_0) > \mu(p_1) > \cdots$, which would contradict that $(D, >)$ is well-founded). The method is also *complete*: if a program terminates, then it has a termination measure (namely, $\mu(p) = p$, because the set of program states, with the relation $p > p'$ if there is a transition from p to p', is a well-founded set).

The practical reason is that termination measures are mostly simple because they reflect an algorithmic approach conceived in the mind of a human. Tied to this is a common tenet that programs should be developed along with, or based on, local descriptions. Thus it is a fundamental to discover to what extent one can rely on local descriptions to deduce global properties.

This subject has been pursued extensively for properties involving fairness, liveness, temporal logic or automata, for example [1,5,6,9,15,17,20,21,32,36,37,46, 50]. Most of the proposed methods are indirect—depending on repeated program transformations in order to reduce the original verification problem to problems that can be solved by standard techniques such as well-founded sets or refinement mappings.

This thesis shows how to verify infinite computations by directly relating pro-

gram and specification. The idea is to measure how finite computations "get closer" to satisfying the specification, in the same way that well-founded sets express approaching termination. We develop the concept of *progress measures*—the main innovation of this thesis. A progress measure μ is used just as a well-founded set is used to show program termination: μ assigns information to each state p of the program P such that on any transition from p to p', it is the case that $\mu(p) \rhd \mu(p')$; here $\rhd$ is a *progress relation*, which in the limit ensures that the specification S is satisfied; that is, if $\mu(p_0) \rhd \mu(p_1) \rhd \cdots$ is an infinite $\rhd$-related sequence, then $p_0, p_1, \ldots \in L(S)$ holds. Thus the existence of a progress measure implies that $L(P) \subseteq L(S)$, because if $p_0, p_1, \ldots$ is in $L(P)$ then $\mu(p_0) \rhd \mu(p_1) \rhd \cdots$ holds by the verification condition.

We formulate several kinds of progress measures. Each represents a sound and complete verification method for a class of specifications, i.e. if a progress measure exists, then the language containment holds and vice versa. To explain these classes, we need to discuss how specifications are related to the Borel hierarchy of program properties.

A *program property* is a set of infinite computations. For example, the program property S1="the lamp stays on all the time" is the set of state sequences for which the lamp is always turned on—assuming some system whose states indicate whether the lamp is on.

Some program properties can be invalidated in a finite amount of time. For example, S1 is invalidated as soon as the lamp goes off at some moment. In contrast, the property S2="the lamp flashes infinitely often" can never be invalidated in any finite amount of time, because the lamp can always possibly flash again.

To distinguish such features of program properties, we use the Borel hierarchy of sets of infinite sequences. The simplest kind of program property is a *safety property*, which expresses that nothing bad happens. Property S1 above is a safety property expressing that the bad thing "the lamp goes off" does not happen. Following [4] we identify safety properties with the closed sets on the natural topology on the space of infinite computations (the Baire space). As discussed in Chapter 4, the class of closed sets, denoted $\mathbf{\Pi^0_1}$ (or $\mathbf{F}$), is exactly the class of sets accepted by *safety automata*, which are automata with bounded nondeterminism. We introduce *ND measures*, which constitute a sound and complete verification method for safety automata specifications. The class $\mathbf{\Pi^0_1}$ is summarized in the first

	Class	*Defined by*	*Examples*	*Measure*
$\mathbf{\Pi_1^0}$	closed sets	safety automata	safety properties	ND
$\mathbf{\Pi_2^0}$	intersections of open sets	Büchi cond.	weak fairness	
	Boolean comb. of $\mathbf{\Pi_2^0}$ sets	Rabin cond.	general fairness	Rabin
$\mathbf{\Sigma_3^0}$	unions of $\mathbf{\Pi_2^0}$ sets	inf. Rabin cond.	extreme fairness	
		Liminf cond.		Liminf
$\mathbf{\Pi_3^0}$	intersections of $\mathbf{\Sigma_2^0}$ sets	¬Liminf cond.		¬Liminf
$\mathbf{\Pi_1^1}$	coanalytic sets	WF cond.	inf. temp. logic	WF

Figure 1.1: Some Borel classes.

row of Figure 1.1.

At the next level of the Borel hierarchy, the class $\mathbf{\Pi_2^0}$ (also called G_δ) consists of the countable intersections of open sets. This class corresponds to *weak fairness constraints* [18] expressing that good things happen infinitely often; property S2 is an example of such a property. The class $\mathbf{\Pi_2^0}$ is exactly the class of sets accepted by deterministic *Büchi automata* [11,30]. A verification method for these automata is formulated in [5].

In practice when reasoning about fairness, one often needs certain combinations of conditions of the form "something happens infinitely often" and "something happens finitely often." Such combinations, called *general fairness constraints* in [17], correspond to the acceptance conditions of *Rabin automata*, which generalize Büchi automata. Deterministic Rabin automata exactly define the class of Boolean combinations of $\mathbf{\Pi_2^0}$ sets. In Chapter 5 we exhibit a progress measure for Rabin conditions and show that there are sound and complete verification methods for deterministic Rabin automata and nondeterministic finite-state Büchi automata. As even the extended temporal logic of [52] can be translated into such Büchi automata, it follows that one can measure progress towards satisfaction of any property defined by a finite temporal formula.

Not all notions of liveness can be captured with Rabin automata. For example, *extreme fairness* [41], which corresponds to infinite Rabin conditions, is at level $\mathbf{\Sigma_3^0}$ (also called $G_{\delta\sigma}$), the class of countable unions of $\mathbf{\Pi_2^0}$ sets. Level $\mathbf{\Pi_3^0}$ properly includes the Boolean combinations of $\mathbf{\Pi_2^0}$ sets. In Chapter 6 we discuss verification

with infinite Rabin conditions.

To obtain a more abstract theory of verification, we introduce the simpler *Liminf condition* in Chapter 7. This condition also defines the class of $\mathbf{\Sigma_3^0}$ properties. We exhibit a progress measure for Liminf conditions together with a sound and complete verification method for automata with Liminf acceptance conditions. Also, we give a measure for negated Liminf conditions.

There are properties, such as those expressible by *infinitary temporal logics* (having countable conjunctions and disjunctions), that define sets arbitrarily high in the Borel hierarchy. To describe such sets we introduce in Chapter 8 the notion of *WF condition*, which in fact define the even larger class $\mathbf{\Pi_1^1}$ of coanalytic sets, and we show that there exist progress measures for WF conditions. We prove that infinitary temporal formulas can be expressed as WF conditions. Thus progress towards satisfaction of very general properties can be measured.

1.1 Thesis Overview

For simplicity of exposition, this thesis is mainly written without references to computer languages or systems; we have even in some chapters preferred to study graphs instead of automata.

Yet in Part I, we shall motivate and explain our work with examples that are typical of verification issues that come up in practice. They illustrate nondeterminism (Chapter 2) and liveness (Chapter 3)—issues that appear over and over in distributed programming.

Part II is the mathematics of progress measures. It can be read independently of Part I. We treat nondeterministic specifications (Chapter 4), finite Rabin conditions (Chapter 5), and infinite Rabin conditions (Chapter 6). Then we study verification with the Liminf condition and its negation, and we establish a connection to descriptive set theory (Chapter 7). Finally an optimal (in a recursion-theoretic sense) verification theorem is presented and the Suslin-Kleene Theorem is derived (Chapter 8).

Part III contains applications. By means of examples, we formulate a new method for proving that a program fairly terminates (Chapter 9). We extend the method to extreme fairness (Chapter 10).

Finally we summarize the thesis (Chapter 11).

1.2 Notation

$\langle a_1, \ldots, a_n \rangle$	the finite sequence $a_1, \ldots, a_n$.
$\langle a_1, a_2, \ldots \rangle$	the infinite sequence $a_1, a_2, \ldots$.
$\cdot$	concatenation of sequences.
$u \propto \alpha$	u is a finite prefix of α.
$\mathcal{P}M$	the set of subsets of M.
$\mathcal{F}M$	the set of finite subsets of M.
M^*	the set of finite sequences of elements of M.
M^ω	the set of infinite sequences of elements of M.
ω	the set of natural numbers $\{0, 1, \ldots\}$; also, the first ordinal that is not finite.
ω^ω	the set of infinite sequences of natural numbers $\{0, 1, \ldots\}$.
$G = (V, E)$	a directed countable graph with $E \subseteq V \times V$.
$G\|U$	the induced subgraph on $U \subseteq V$.
E^*	the transitive closure of E.
$v \rightarrow_U^* v'$	there is some path $v_0, \ldots, v_n$ in $G\|U$ with $v_0 = v, v_n = v'$.
$v \rightarrow_U^* W$	for every v' in W, $v \rightarrow_U^* v'$ holds.
$\exists^\infty p(i)$	$p(i)$ holds for infinitely many values of $i \in \omega$, i.e. $\forall K : \exists k > K : p(k)$.
$\forall^\infty p(i)$	$p(i)$ holds for all but finitely many values of i, i.e. $\exists K : \forall k > K : p(k)$.
$\mathrm{dom} f$	domain of function f.
$\mathrm{rng} f$	range of function f.

Notation for automata is provided in Section 4.1 and for trees in Section 5.2.

Part I

Issues

Chapter 2

Reasoning with Nondeterminism

Nondeterminism is the subject of this chapter. Using a rather mundane example, we argue how nondeterminism arises naturally in both program and specification. We show that relating nondeterministic descriptions is more intriguing than one would think at first.

2.1 Grammars for Popular Songs

Although nondeterminism sounds scientific, it shows up even in the humanities. For example, popular songs can be analyzed using grammar-like descriptions such as $ABA'A$ or $ABCA$ (as in the book accompanying the Smithsonian collection "American Popular Song" [7].) Of course, when listening to a song the first time, one never knows after having heard A whether $BA'A$ or BCA follows; in this sense these descriptions are nondeterministic. We hypothesize here that analysis of popular songs can take place using nondeterministic grammars.

Now consider a musicologist who has developed a grammar S specifying carefully tested socio-psychological principles for how sequences of pop songs are best put together for a light-music radio station. The musicologist is faced with the problem of verifying that a proposed expert system program P for automatically producing popsongs only emits sequences of songs described by S. Since the radio station broadcasts endlessly, there is no natural bound on how many songs should be considered before comparing the output of program P with specification S. Instead the musicologist considers the limit, that is, the infinite sequences of songs that are produced if the system is run forever.

Looking at these infinite objects, however, is usually not the best way to proceed. Often there will be uncountably many of them. Instead one would like to relate states of P and S in a way that guarantees that P satisfies S; in other words, a verification method is needed.

2.2 Relationships between System and Description

To help the musicologist find a verification method, we define how P and S work as *automata*, also called *transition systems*. A computation of P produces a sequence $\alpha = \text{♩}_0, \ldots, \text{♩}_n, \bullet, \text{♩}, \ldots, \text{♩}, \bullet, \ldots$ of notes ♩ interspersed with bullets $\bullet$, each bullet denoting the end of a song. Thus the first song of α is $\text{♩}_1, \ldots, \text{♩}_n$, where each ♩ denotes the various kind of information that constitute a note, for example pitch and duration. The sequence α is called a *behavior*.

P is given by a set of states, where p^0 is the *initial state*, and a transition relation $\rightarrow_P$. A *transition* $p \rightarrow_P p'$ takes P from a state p to a new state p' while emitting a symbol e, which is a ♩ or a $\bullet$. A *computation* (or *run*) of P over a behavior $e_0, e_1, \ldots$ is a sequence of states $p_0, p_1, \ldots$ such that p_0 is the initial state p^0 and $p_0 \xrightarrow{e_0} p_1 \xrightarrow{e_1} \cdots$. The transition relation is *nondeterministic* if there are p, p', p'', and e such that $p' \neq p''$, $p \xrightarrow{e} p'$, and $p \xrightarrow{e} p''$. In that case one cannot tell, even knowing p and e, what the new state is. We assume that P is nondeterministic so that different radio stations may send different sequences of songs.

The grammar S can also be described as a transition system with an initial state s^0 and a transition relation $\rightarrow_S$. For example, it is natural to describe the grammar S by a nondeterministic pushdown automaton [23], which we also denote by S.

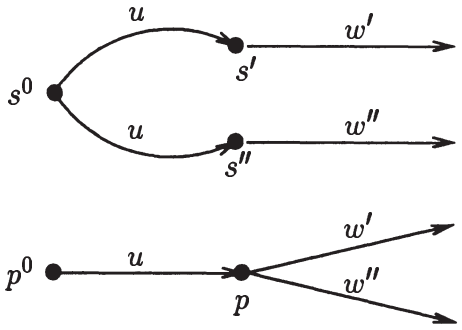
Next, to define a verification method, we consider the simple idea of mapping each state of P into a state of S. This approach is plausible, for whenever there is a computation of P over some behavior α, there must be a corresponding behavior of S over α. More precisely, the method consists of finding a mapping μ, called a *refinement measure*, that satisfies two criteria. It must map the initial state of P to the initial state of S, i.e. $\mu(p^0) = s^0$. Moreover, for every transition $p \xrightarrow{e}_P p'$ of P, there must be a transition $\mu(p) \xrightarrow{e}_S \mu(p')$ of S. These two conditions are *verification conditions* because they are local, i.e. about states and single transitions. It is not

hard to see that they imply that any behavior of P is a behavior of S. In fact, let $e_0, e_1, \ldots$ be a behavior of P. Then there is a computation $p_0 \xrightarrow{e_0}_P p_1 \xrightarrow{e_1}_P \cdots$ of P, and from the verification conditions it follows that $\mu(p_0) \xrightarrow{e_0}_S \mu(p_1) \xrightarrow{e_1}_S \cdots$ is a computation of S.

2.3 Incompleteness of Simple Measures

Unfortunately the method just outlined is not complete; that is, even if P satisfies S, a refinement measure may not exist. To see this, let us denote by $p \bullet\!\!\xrightarrow{u}\!\!\bullet\ p'$ that there are $p_0, \ldots, p_n$ such that $p = p_0 \xrightarrow{e_0}_P \cdots \xrightarrow{e_n}_P p_{n+1} = p'$, where $u = e_0, \ldots, e_n$ is a finite behavior. Similarly, $p \bullet\!\!\xrightarrow{w}$ means that there are $p_0, p_1, \ldots$ such that $p = p_0 \xrightarrow{e_0}_P p_1 \xrightarrow{e_2}_P \cdots$, where $w = e_0, e_1, \ldots$ is an infinite behavior. A state p is *reachable* over u if there is some p such that $p^0 \bullet\!\!\xrightarrow{u}\!\!\bullet\ p$. Now assume that P satisfies S and that this can be proved by a refinement measure μ.

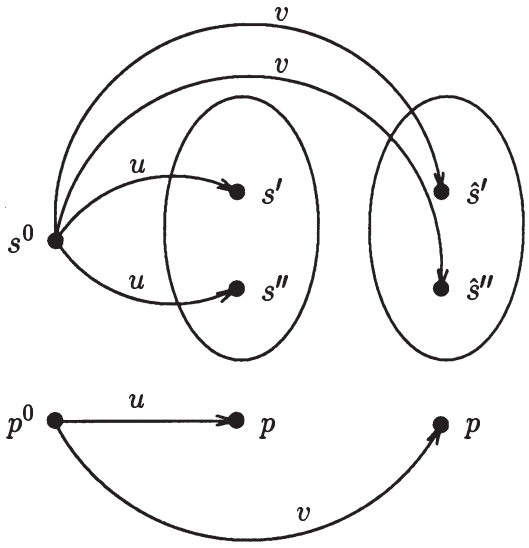
Consider the situation:



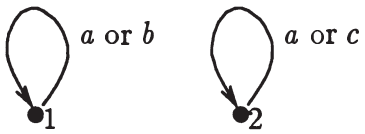
where states p of P and s', s'' of S are all the states reachable over some u. Also there are w' and w'' such that $u \cdot w'$ and $u \cdot w''$ are different behaviors. Suppose that $p_0, \ldots, p, p'_0, p'_1, \ldots$ is a computation of P over $u \cdot w'$ and that $p_0, \ldots, p, p''_0, p''_1, \ldots$ is a computation over $u \cdot w''$. Then $\mu(p_0), \ldots, \mu(p), \mu(p'_0), \mu(p'_1), \ldots$ must be a computation of S over $u \cdot w'$ and $\mu(p_0), \ldots, \mu(p), \mu(p''_0), \mu(p''_1), \ldots$ must be a computation of S over $u \cdot w''$. But this is impossible, since for $u \cdot w'$, it must be the case that $\mu(p) = s'$, and for $u \cdot w''$, it must be the case that $\mu(p) = s''$. Thus the measure $\mu(p)$ should be a set, called a *prophecy set*, of states of S, and it should satisfy the verification condition: $p \xrightarrow{e}_P p'$ implies $\mu(p) \overset{e}{\triangleright}_S \mu(p')$, where $\triangleright_S$ is some *progress relation*. The essential property of this relation is that from

$\mu(p_0) \overset{e_0}{\triangleright}_S \mu(p_1) \overset{e_1}{\triangleright}_S \cdots$, one must be able to extract a computation of S.

Even if a method based on mapping program states to sets of specification states is found, it still cannot be complete. Informally, the problem is that a state p of P may be reached by many different behaviors, requiring the method to distinguish among different prophecy sets; two such sets, corresponding to finite behaviors u and v, are marked below:



To see more formally that no method exists, consider the transition system S:



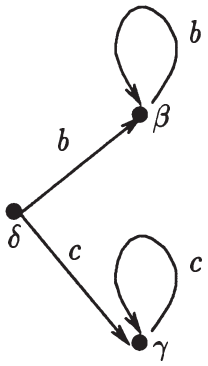
where both states 1 and 2 are initial states. The behaviors defined by S are the sequences that consist of either a's and b's or a's and c's (i.e. the ω-regular language $(a|b)^\omega \cup (a|c)^\omega$). We show that there can be no relation $\triangleright_S$ such that a verification method using measures based on $\triangleright_S$ is both sound and complete. We assume that such a method would satisfy two criteria:

i. If $C_0 \overset{e_0}{\triangleright}_S C_1 \overset{e_1}{\triangleright}_S C_2 \overset{e_2}{\triangleright}_S \cdots$, where $C_0, C_1, \ldots$ are sets of specification states, then there are $s_0 \in C_0$, $s_1 \in C_1, \ldots$ such that $s_0 \overset{e_1}{\rightarrow}_S s_1 \overset{e_2}{\rightarrow}_S \cdots$ is a computation of S.

ii. A state s need only be in $\mu(p)$ if there is a u such that both p and s are reachable over u and for some infinite behavior w, $u \cdot w$ is allowed by S.

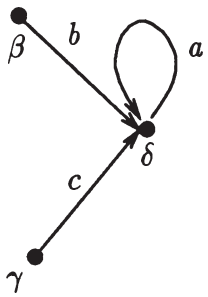
The justification for Criterion ii is that a reasonable method must allow a measure μ such that $\mu(p)$ only contains states that occur in computations of S when p occurs in a corresponding computation of P.

Our proof involves two programs. The first is $P1$:



where state δ is the initial state. There are two infinite behaviors of $P1$, namely $b, b, b, \ldots$ and $c, c, c, \ldots$, i.e. $P1$ satisfies S. Thus since we assume that the hypothesized method is complete, there is a measure μ. By Criterion i, $\mu(\delta)$ must contain state 1, because $\delta \xrightarrow{b}_{P1} \beta \xrightarrow{b}_{P1} \beta \xrightarrow{b}_{P1} \cdots$ is a computation of $P1$, and the only corresponding computation of S is $1 \xrightarrow{b}_{S} 1 \xrightarrow{b}_{S} \cdots$. Similarly, $\mu(\delta)$ must contain 2, $\mu(\beta)$ must contain 1, and $\mu(\gamma)$ must contain 2. By Criterion ii, $\mu(\beta)$ does not contain 2, and $\mu(\gamma)$ does not contain 1. Thus $\mu(\delta) = \{1, 2\}$, $\mu(\beta) = \{1\}$, and $\mu(\gamma) = \{2\}$. Since $\delta \xrightarrow{b}_{P1} \beta \xrightarrow{b}_{P1} \beta \xrightarrow{b}_{P1} \cdots$ is a computation of $P1$, it must hold that $\mu(\delta) \overset{b}{\triangleright}_S \mu(\beta) \overset{b}{\triangleright}_S \mu(\beta) \overset{b}{\triangleright}_S \cdots$; in particular, $\{1,2\} \overset{b}{\triangleright}_S \{1\}$. By an analogous argument, $\{1,2\} \overset{c}{\triangleright}_S \{2\}$ must hold.

The second program is $P2$:



whose initial states are β and γ. The behaviors of this program are $b, a, a, a, \ldots$

and $c, a, a, a, \ldots$ Thus $P2$ satisfies S. By arguments similar to those above, it can be seen that $\mu(\beta) = \{1\}$, $\mu(\gamma) = \{2\}$, and $\mu(\delta) = \{1,2\}$. Thus $\{1\} \overset{b}{\triangleright}_S \{1,2\}$ and $\{2\} \overset{c}{\triangleright}_S \{1,2\}$ must hold.

From these sample programs, we conclude that there is a sequence $\{1,2\} \overset{c}{\triangleright}_S \{2\} \overset{c}{\triangleright}_S \{1,2\} \overset{b}{\triangleright}_S \{1\} \overset{b}{\triangleright}_S \{1,2\} \overset{c}{\triangleright}_S \cdots$, but this contradicts Criterion i, because S does not allow the behavior $c, c, b, b, c, c, b, b, \cdots$.

We might therefore consider $\mu(p)$ to be a set of sets of states of S. In Chapter 3 we show that on such mappings, it is possible to define relations that yield a sound and complete verification method under the restriction that S have bounded nondeterminism.

Chapter 3

Reasoning with Liveness

This chapter is an introduction to the concepts of progress measures for *Rabin conditions*. Such conditions can express many liveness properties about the repetition of states, for example when describing fair executions. We sketch the ideas behind *Rabin measures*, which are progress measures for Rabin conditions. These progress measures are formally treated in Chapters 5 and 6. As it is shown in Chapters 9 and 10, they open the way for extending assertional methods for program verification to handle fairness constraints without the introduction of program transformations.

The chapter consists of three sections. In the first section we use the example of a pop song computer from Chapter 2 to motivate the need for expressing the repeated occurrence of events. In the second section we consider the same system, changed so that it is now in a network where its disk can be used by other computers. This system is susceptible to *livelock*—a situation that occurs when a process or computer from some point onward is never granted access to a resource, such as a disk, even though access is infinitely often requested or possible. We show how to reason about the system with this complication. In the last section we discuss some simple mathematical means of expressing more general liveness properties.

3.1 A Persistent Need For Pop Songs

The need for pop songs is unlimited. It is therefore relevant to verify that the pop song producing computer P from Chapter 2 outputs songs over and over. We call this requirement the $\star$-*condition*, and to analyze it, we take a more detailed view of P. A transition may either produce a note or produce nothing—the latter

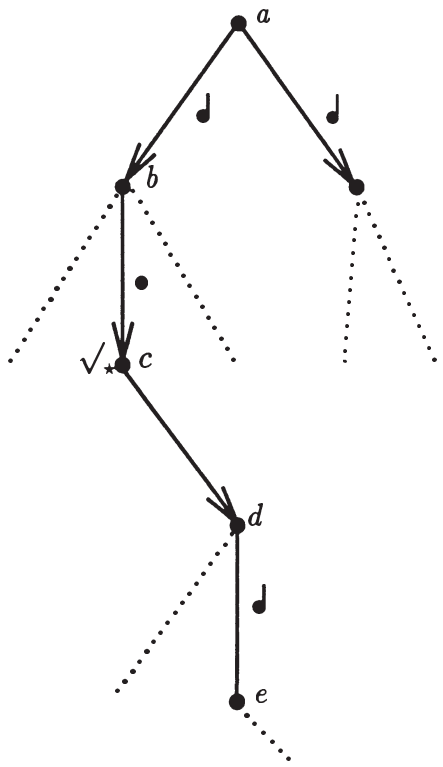


Figure 3.1: Computation graph G of P.

happening when the system is in a livelock situation, for example. To simplify things, we focus here on infinite state sequences instead of behaviors. We view P as a (possibly infinite) graph $G = (V, E)$; see Figure 3.1. The set of vertices V represents states of P, and the vertex a at the top of the diagram corresponds to the initial state. The set of edges $E \subseteq V \times V$ corresponds to the transitions of P. An edge (u, v), from u to v, is marked with ♩, •, or nothing, depending on which, if any, symbol is emitted. We mark vertices that correspond to the end of a pop song by a checkmark $\surd_\star$. For example, since the transition from vertex b to vertex c emits a • in Figure 3.1, vertex c is marked. We let $R_\star$ designate the set of vertices marked with $\surd_\star$.

Computations of P correspond to infinite paths $v_0, v_1, \ldots$ in G starting at the top vertex. Therefore we say that an infinite path $v_0, v_1, \ldots$ *satisfies* the $\star$-condition and write $v_0, v_1, \ldots \models R_\star$ if there are infinitely many v_i marked with $\surd_\star$. Here the set $R_\star$ constitutes a simple example of a *Rabin condition*. The graph G satisfies

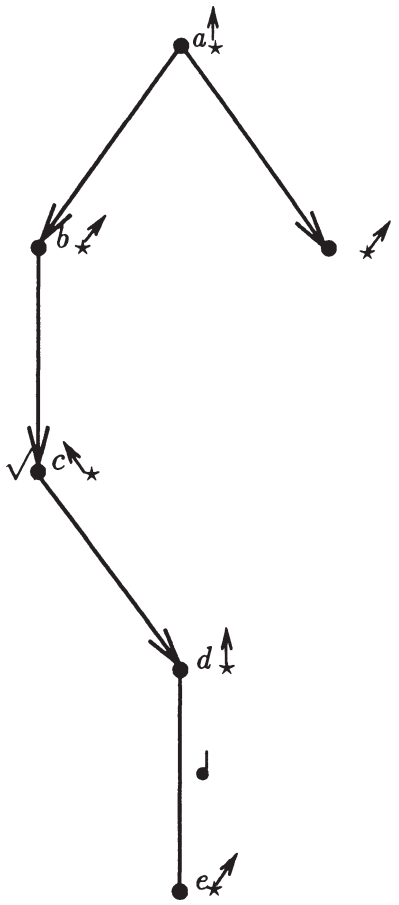


Figure 3.2: G with measure.

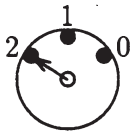
$R_\star$, written $G \models R_\star$, if all infinite paths of G contain infinitely many vertices in $R_\star$.

Expressing $G \models R_\star$ in logic involves a universal quantification over infinite objects. It is, however, well known that determining $G \models R_\star$ instead can take place by associating some information with each vertex of G and checking that some verification condition is satisfied. It suffices to find a set D with a well-founded ordering $>$ and a mapping μ that associates an element $\mu(v)$ of D to each vertex v such that the following verification condition holds:

(V) For every transition (u, v), either $v \in R_\star$ or $\mu(u) > \mu(v)$.

Figure 3.2 illustrates how this can be done. Each value of μ is given as a *pointer*. The use of pointers is essential to our method of verifying with Rabin conditions; it also gives a nice visual explanation, since a pointer moves as on the

dial of a countdown clock:



which ticks off to the next occurrence of a state in $R_\star$. These pointers are called $\star$-pointers since they pertain to the $\star$-condition. On the transition from a to b, the $\star$-pointer, i.e. the value of μ, decreases from 1 to 0, in accordance with (V). On the transition from b to c, however, the value of μ increases from 0 to 2, but (V) is still fulfilled, because c is in $R_\star$; in this case the clock is "rewound."

The mapping μ is a *progress measure* for $(G, R_\star)$, because for each edge of G, it expresses headway towards satisfaction of $R_\star$. In fact one can prove:

$G \models R_\star$ if and only if $(G, R_\star)$ has a progress measure;

that is, the method of measures is sound ("if") and complete ("only if") for proving that G satisfies $R_\star$.

3.2 The Pop Song Example—Networked Version

Livelock is an intricate problem that often arises in distributed systems. In our example it might occur if the computer P is connected to a network of other computers and it is allowed to release its disk to the network whenever it has finished a song so that other computers may use the disk.

Livelock occurs when the disk is used over and over by other computers, preventing P from regaining control. Often one can assume this will not happen in practice, either for probabilistic reasons, or because when other computers use the disk, there is some feedback to P enabling it to recapture its disk for its own use. Given this assumption, how can one prove that P still produces infinitely many pop songs?

To answer this we use $\surd_\lambda$ to mark states in which the disk is used by another computer; here λ stands for "livelock." In Figure 3.3 vertices f and g represent the choice that after a pop song has been finished at vertex d, either P can release the disk at vertex f or proceed with another song at vertex e. At vertex d there might be an arbitrary number of outstanding requests for using the disk. Thus the disk release routine (d, f, and g) is a loop. But when P has committed itself to

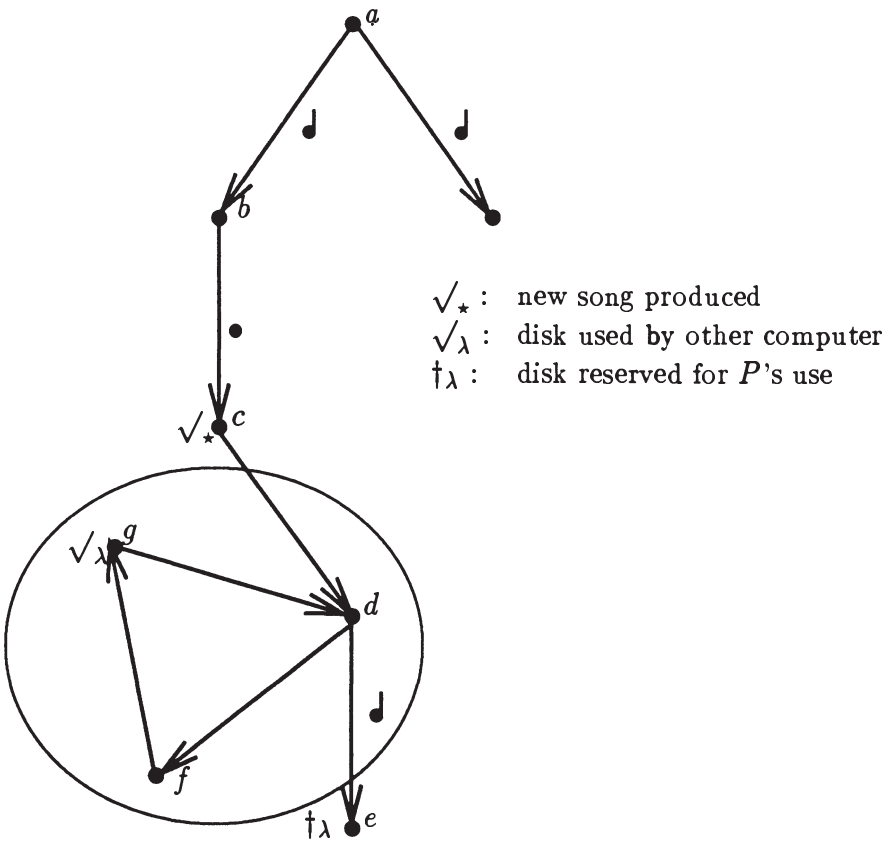


Figure 3.3: G containing a livelock situation.

produce another pop song, it enters state e, which is marked with a $\dagger_\lambda$ to denote that the disk gets reserved for P's use; here the dagger symbolizes the "death" of the liveness supposition.

Livelock occurs when other computers use the disk over and over while from some point on the disk never gets reserved for P's own use. Thus an infinite path exhibits livelock when there are only finitely many vertices marked $\dagger_\lambda$ and infinitely many marked $\surd_\lambda$.

We let R_λ be the set of vertices marked with $\surd_\lambda$, and we let I_λ be the set of vertices marked with $\dagger_\lambda$. The pair (R_λ, I_λ) is called a *Rabin pair* and also referred to as the *λ-condition*. When $v_0, v_1, \ldots$ exhibits livelock, i.e. when infinitely often $v_i \in R_\lambda$ and only finitely often $v_i \in I_\lambda$, we say that $v_0, v_1, \ldots$ satisfies (R_λ, I_λ), written $v_0, v_1, \ldots \models (R_\lambda, I_\lambda)$. For example, any path that ends $d, f, g, d, f, g, \ldots$ satisfies (R_λ, I_λ).

Proving that infinitely many pop songs are produced under assumption of no

livelock is the same as proving that any infinite path in G satisfies either $R_\star$ or (R_λ, I_λ). (We have used the propositional equivalence $(\neg L \Rightarrow S) \equiv (L \vee S)$.) The disjunction "$R_\star$ or (R_λ, I_λ)" is an example of a *Rabin condition*.

Whenever there is progress towards producing a song, we can use the $\star$-pointers from the last example. But what do we do in the loop d, f, g? Here there is no progress towards producing another pop song—indeed, this is a livelock situation. Thus at this point we say that the $\star$-condition is a *dormant hypothesis*, and we use the pointer $\updownarrow_\star$ of d all the way around d, f, g. The lack of progress towards the $\star$-condition is compensated for, however, if we hypothesize that the λ-condition is the one that is going to be fulfilled. The progress towards this condition can be measured by using λ-pointers, and we say that the *λ-hypothesis* is *active*. Where to put the λ-pointers?

The solution, which exposes the key idea of Chapter 5, is to put them on top of the $\star$-pointers; see Figure 3.4. Progress from vertex to vertex is checked as before. From d to f the λ-pointer decreases. From f to g a vertex in R_λ is encountered and the λ-pointer is rewound. Note, however, that it is not possible to sustain the λ-hypothesis from d to e, since at e the livelock situation does not exist anymore; instead the $\star$-hypothesis is active and progress is measured using the $\star$-pointer.

The ideas presented here work for any number of Rabin pairs; for a formal treatment, see Chapter 5 (finite Rabin conditions) and Chapter 6 (countable Rabin conditions). In Chapters 9 and 10, which both can be read independently of the formal treatment, we take up another problem that can be solved using Rabin measures, namely fair termination.

3.3 General Liveness Properties

Is it possible to measure progress towards satisfaction of more complicated properties than those we have just seen? One might think that the mathematics of such a measure would get too complicated. In fact the opposite holds if one adopts some mathematically simple techniques for specifying infinite computations. Instead of marking vertices of a graph with $\surd$'s and †'s, we assign a natural number $h(v)$, called a *height*, to each vertex v; see Figure 3.5. The mapping h constitutes a specification, called a *Liminf specification*. An infinite path $v_0, v_1, \ldots$ satisfies the *Liminf condition* if $\liminf_{i\to\omega} h(v_i)$ is finite, i.e. if $h(v_i)$ dips down to some fixed

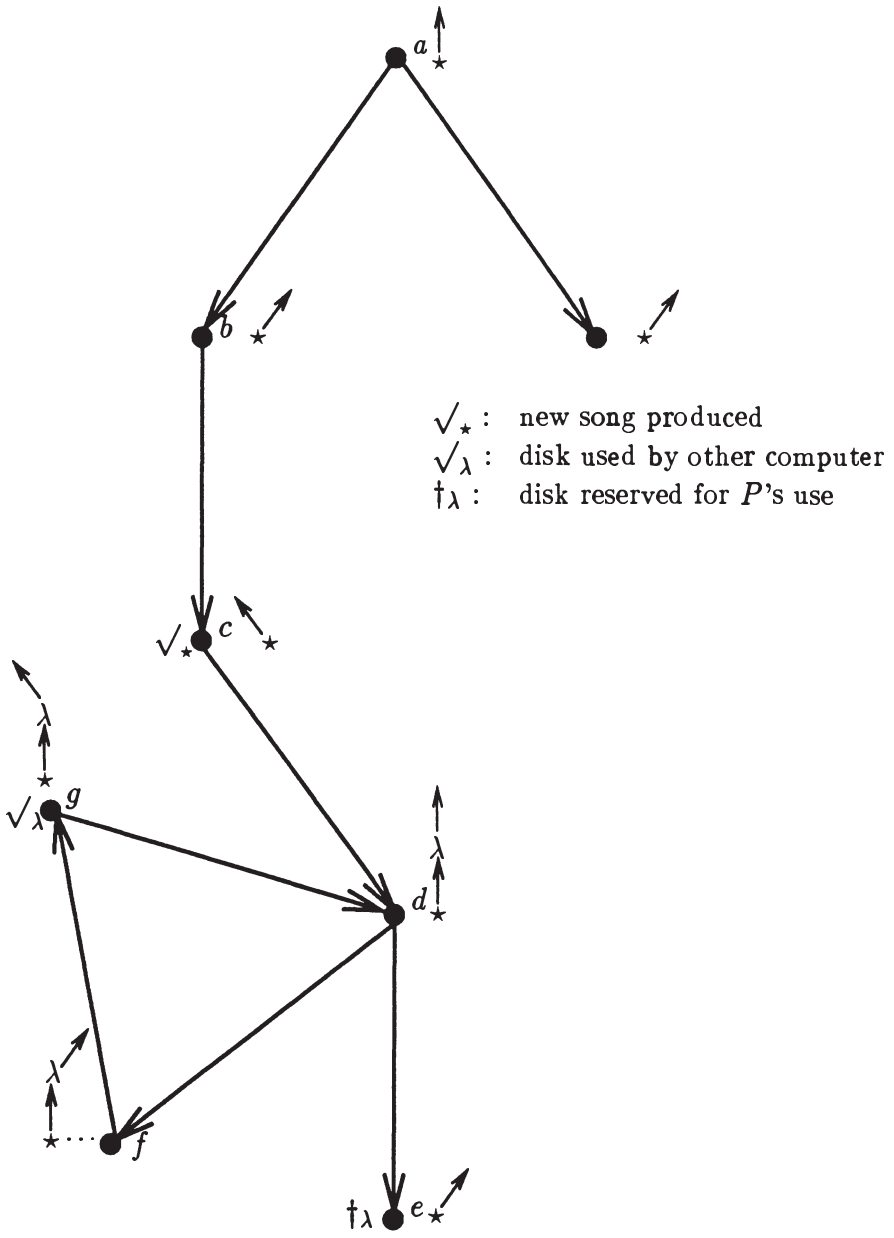


Figure 3.4: G with livelock and Rabin measure.

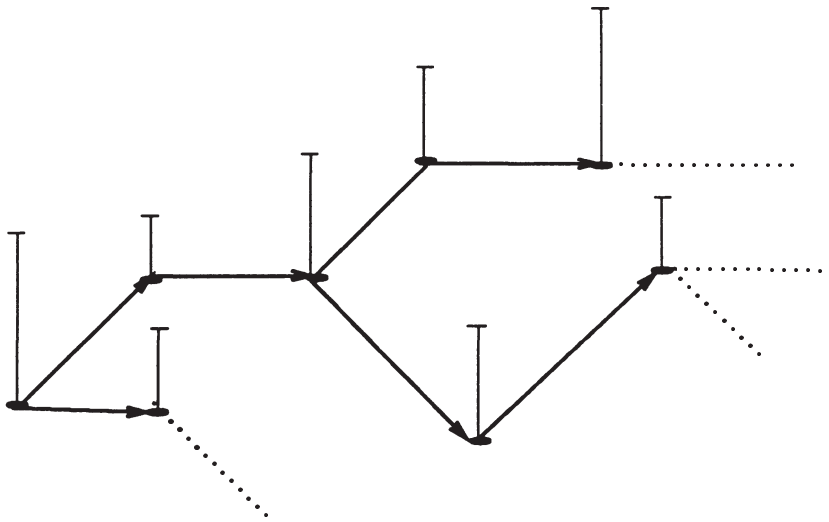


Figure 3.5: A graph with a height assignment.

value infinitely often as i goes towards infinity. One can specify many liveness properties with such a specification. For a trivial example, the condition that infinitely often a vertex marked $\surd_\star$ is encountered can be coded by setting the height of every vertex marked with $\surd_\star$ to zero and the height of an unmarked vertex to the height of the preceding vertex plus one (assuming that each vertex has one predecessor). If an infinite path $v_0, v_1, \ldots$ has infinitely many vertices marked $\surd_\star$, then $\liminf h(v_i) = 0$; otherwise if there are only finitely many marked vertices, then $\liminf h(v_i)$ is infinite. All finite temporal logic formulas and all Rabin conditions, finite or countable, can be coded in this way when G is a tree (which it is if vertices represent finite computations; see Chapter 7).

Can it be checked locally, using a progress measure, that a graph G satisfies a Liminf condition h, i.e. that for all paths $v_0, v_1, \ldots$ in G, $\liminf h(v_i)$ is finite? The answer is affirmative as explained in Chapter 7; it is possible to assign information to each vertex and check for each transition that progress towards reaching a finite liminf is made.

This abstract method of verification is generalized in Chapter 8 to assignments of finite trees to the vertices of G. Very general specifications can be expressed using such assignments, including infinitary temporal logics. Thus in theory, one can always measure headway toward satisfaction of even very complicated temporal specifications.

Part II

Solutions

Chapter 4

Safety automata

For nondeterministic automata A_P and A_S defining sets of infinite behaviors $L(A_P)$ and $L(A_S)$, we study the problem of determining whether A_P *implements* or *satisfies* A_S, i.e. whether every behavior of A_P is a behavior of A_S or equivalently whether $L(A_P) \subseteq L(A_S)$. We show how $L(A_P) \subseteq L(A_S)$ can be determined using a progress measure, called the *ND measure*, which relates A_P and A_S; the method is complete, however, only if A_S is a *safety automaton*—this means that its nondeterminism is bounded (finite). Many automata, including those with finitely many states, satisfy this criterion. Safety automata define precisely the class $\mathbf{\Pi_1^0}$ of closed sets, which are regarded by computer scientists as *safety properties*.

To explain the role of an ND measure, we call A_P the *program automaton* (and its states the *program states*); similarly A_S is the *specification automaton* (and its states are *specification states*). An ND measure μ is a mapping that to each program state p associates a set $\mu(p)$ of sets of specification states; moreover μ must respect the *ND relation*, which is the main innovation of this chapter. The essential property of the ND measure is: from an infinite ND-related sequence one can obtain a run of the specification automaton.

This chapter is organized as follows. We give definitions of automata in Section 4.1. In Section 4.2, which is independent of the rest of the chapter, we relate safety automata and safety properties. We give our main results about ND measures in Section 4.3. In Section 4.4 these results are used to derive previous methods. We discuss our approach in Section 4.5 in terms of recursion theory, and in Section 4.6 we provide an overview of related work.

4.1 Definitions

Let Σ be a fixed (at most) countable alphabet of symbols called *events* (representing e.g. actions, communications, observable parts of states). A *behavior* is a sequence (infinite if not otherwise stated) $e_0, e_1, \ldots$ of events. Let V be a set of *states*. A *transition relation on* V is a relation $\rightarrow \subseteq V \times \Sigma \times V$; a transition $(v, e, v') \in \rightarrow$ is denoted $v \xrightarrow{e} v'$. An *automaton* $A = (\Sigma, V, \rightarrow, V^0)$ consists of an alphabet Σ, a state space V, a transition relation $\rightarrow$ on V, and a set of *initial states* $V^0 \subseteq V$. A *run* (computation) of A over a behavior $e_0, e_1, \ldots$ is an infinite $\rightarrow$-related sequence of states $v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots$ with $v_0 \in V^0$. A behavior $e_0, e_1, \ldots$ is *accepted by* A—or is a *behavior of* A—if there is a run of A over $e_0, e_1, \ldots$ The *language* or *property* $L(A)$ *accepted by* A is the set of behaviors of A.

A *partial run* over a finite behavior $e_0, \ldots, e_n$ is a finite $\rightarrow$-related sequence of states $v_0 \xrightarrow{e_0} \cdots \xrightarrow{e_n} v_{n+1}$, where $v_0 \in V^0$. A state v is *reachable* over $e_0, \ldots, e_n$ if there is a partial run $v_0 \xrightarrow{e_0} \cdots \xrightarrow{e_n} v_{n+1}$ with $v = v_{n+1}$. The set of states reachable over $e_0, \ldots, e_n$ is denoted $\mathcal{R}_A(e_0, \ldots, e_n)$. Note that $\mathcal{R}_A() = V^0$.

A transition relation $\rightarrow$ has *bounded nondeterminism* if for all $e \in \Sigma$ and all $v \in V$, the set $\{v' \mid v \xrightarrow{e} v'\}$ is finite. Automaton $A = (\Sigma, V, \rightarrow, V^0)$ is a *safety automaton* if V^0 is finite and $\rightarrow$ has bounded nondeterminism. In particular, if V^0 and all sets $\{v' \mid v \xrightarrow{e} v'\}$ have at most one element, then A is *deterministic*. Note that this implies that for all $e_0, \ldots, e_n$, the set $\mathcal{R}_A(e_0, \ldots, e_n)$ has at most one element. If for all v in V there is at most one sequence $e_0, \ldots, e_n$ such that $v \in \mathcal{R}_A(e_0, \ldots, e_n)$, then A is *historical*; the intuition is that each state corresponds to at most one finite behavior or *history* leading up to that state.

Automaton A is *complete* if $V \neq \emptyset$ and every state $v \in V$ is in some run. Note that such an automaton accepts a non-empty language; moreover, to every automaton $A = (\Sigma, V, \rightarrow, V^0)$ such that $L(A) \neq \emptyset$, one can obtain a complete automaton A' such that $L(A') = L(A)$ by deleting from V states that are not in any run. This property, however, is not decidable, since it essentially involves deciding whether there is an infinite path from a node in a graph—something which is Σ^1_1-complete for countable recursive graphs.

4.2 Safety Properties

By definition *safety properties* are sets of behaviors that are closed with respect to the natural topology on the space Σ^ω (which is the Baire space if Σ is infinite) [4]. Thus a set $S \subseteq \Sigma^\omega$ is a safety property (i.e. a closed set) iff every infinite behavior $e_0, e_1, \ldots$ not in S has some finite prefix u—the "bad thing"—such that no infinite extension $v \in \Sigma^\omega$ makes $u \cdot v \in S$ hold, i.e. such that nothing can undo the bad thing. There is a well-known automata-theoretic characterization of safety properties [11]:

Proposition 1 The following are equivalent:

(a) S is a safety property.

(b) $S = L(A)$ for some safety automaton A.

(c) $S = L(A)$ for some deterministic automaton A.

Proof "(a) $\Rightarrow$ (b)" Consider a safety property $S = \overline{\bigcup_i u_i \cdot \Sigma^\omega}$, where the u_i's are in Σ^* and $\overline{}$ denotes complementation in Σ^ω. Let A be the automaton $(\Sigma, \Sigma^* \backslash \{u_i | i \geq 0\}, \rightarrow, \{\langle\rangle\})$, where $u \xrightarrow{e} v$ iff $u \cdot \langle e \rangle = v$. This defines a deterministic automaton that checks that its current state—the past sequence of events—is not among the u_i's. Thus, $w \in L(A)$ iff w does not have a prefix that is in $\{u_i | i \geq 0\}$ iff $w \in S$.

"(b) $\Rightarrow$ (c)" A construction similar to the classical subset construction for finite-state automata [23] can be applied to obtain a deterministic automaton A_d from a safety automaton $A = (\Sigma, V, \rightarrow, V^0)$ such that $L(A) = L(A_d)$. Let the deterministic automaton A_d be $(\Sigma, \mathcal{F}V, \rightarrow_d, \{V^0\})$, where $M \xrightarrow{e}_d N$ if $N = \{v' \,|\, \exists v \in M : v \xrightarrow{e} v'\} \neq \emptyset$. Note that N is uniquely defined (as a function of M and e) and is finite, because $\rightarrow$ has bounded nondeterminism. We show that $L(A) = L(A_d)$.

$L(A) \subseteq L(A_d)$: Let $v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots$ be a run of A. Let $M_0 = V^0$ and let $M_1 = \{v' \,|\, \exists v \in M_0 : v \xrightarrow{e} v'\}$. Since $v_0 \in M_0$ and $v_0 \xrightarrow{e_0} v_1$, the set M_1 is nonempty; thus $M_0 \xrightarrow{e_0} M_1$. Continuing this argument, we obtain the run $M_0 \xrightarrow{e_0}_d M_1 \xrightarrow{e_1}_d \cdots$ of A_d.

$L(A_d) \subseteq L(A)$: Let $M_0 \xrightarrow{e_0}_d M_1 \xrightarrow{e_1}_d \cdots$ be a run of A_d. Construct a forest as follows. Each node is of the form $v_0 \xrightarrow{e_0} \cdots \xrightarrow{e_{n-1}} v_n$ such that $v_i \in M_i$, for $i \leq n$, and $v_i \xrightarrow{e_i} v_{i+1}$, for $i < n$; in particular, the roots are the elements of $M_0 = V^0$. The edges are of the form

$(v_0 \xrightarrow{e_0} \cdots \xrightarrow{e_{n-1}} v_n\,,\; v_0 \xrightarrow{e_0} \cdots \xrightarrow{e_n} v_{n+1})$.

Since M_i is finite, the forest is a finite collection of finitely branching trees. Since M_i is nonempty, the forest is infinite. Hence by König's Lemma, there is an infinite path through one of the trees. This path defines a run $v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots$ of A.

"(c) $\Rightarrow$ (a)" Let A be a deterministic infinite-state automaton, and let the set $\{u_i \mid i \geq 0\}$ be the set of strings u_i in Σ^* such that there is no partial run of A along u_i. Let $S = \overline{\bigcup_i u_i \cdot \Sigma^\omega}$. By construction, S is a safety property. We show that $L(A) = S$.

$L(A) \subseteq S$: If w is accepted by A, then from the definition of $\{u_i | i \geq 0\}$, w can contain no prefix from among the u_i's. Therefore, $w \in S$.

$S \subseteq L(A)$: Let w be in S. Then by definition of S, there are partial runs of A along all prefixes of w. Because the automaton is deterministic, these partial runs are ordered under the prefix relation (for any two partial runs, one is a prefix of the other). Their limit defines a run over w. Hence w is accepted by A. □

4.3 Measures for Nondeterministic Automata

We now turn to the question of establishing $L(A_P) \subseteq L(A_S)$ without considering infinite runs. Our solution can best be understood as a combination of two previous methods of relating automata, namely those of *prophecy mappings* and *possibilities mappings*, here called *prophecy measures* and *history measures*.

As it will be shown, the method of prophecy measures is complete when the program automaton is historical, i.e. when no state can be reached in more than one way. On the other hand, the method of history measures is complete when the specification automaton is deterministic. To overcome these restrictions we define the ND measure and show that these previous methods follow as corollaries of our approach.

In the following, program automaton A_P is denoted $(\Sigma, V_P, \rightarrow_P, V_P^0)$ and specification automaton A_S is denoted $(\Sigma, V_S, \rightarrow_S, V_S^0)$; thus a program state p is in V_P and a specification state s is in V_S.

4.3.1 Refinement Measure

We start by studying the simple method of refinement measures, which map states of A_P to states of A_S:

Definition 1 A *refinement measure* μ for (A_P, A_S) is a mapping $\mu : V_P \to V_S$ such that:

(REμ1) $p \in V_P^0 \Rightarrow \mu(p) \in V_S^0$

(REμ2) $p \xrightarrow{e}_P p' \Rightarrow \mu(p) \xrightarrow{e}_S \mu(p')$

This method is always sound, i.e. if there is a refinement measure for (A_P, A_S) then $L(A_P) \subseteq L(A_S)$. As discussed in Section 2.3, it is not complete. Only by imposing restrictions on A_P and A_S can one obtain a complete method:

Proposition 2 Let A_P be an historical automaton (assumed complete according to the discussion in Section 4.1) and A_S a deterministic automaton. Then $L(A_P) \subseteq L(A_S)$ iff (A_P, A_S) has a refinement measure.

"$\Leftarrow$" (Soundness) Assume that (A_P, A_S) has a refinement measure μ. By (REμ1) and (REμ2), there is for every run $p_0 \xrightarrow{e_0}_P p_1 \xrightarrow{e_1}_P \cdots$ of A_P a corresponding run $\mu(p_0) \xrightarrow{e_0}_S \mu(p_1) \xrightarrow{e_1}_S \cdots$ of A_S. Thus $L(A_P) \subseteq L(A_S)$.

"$\Rightarrow$" (Completeness) Define $\mu(p) = s$, where $\{s\} = \mathcal{R}_{A_S}(e_0, \ldots, e_n)$ and $e_0, \ldots, e_n$ is the unique finite behavior such that $p \in \mathcal{R}_{A_P}(e_0, \ldots, e_n)$. This definition makes sense because

- by the assumption that A_P is complete, there is a sequence $e_0, \ldots, e_n$, and by assumption that A_P is historical, this sequence is unique; and
- since A_P is complete, there are $e_{n+1}, e_{n+2}, \ldots$ such that $e_0, e_1, \ldots \in L(A_P)$, whence, by the assumption that $L(A_P) \subseteq L(A_S)$, $\mathcal{R}_{A_S}(e_0, \ldots, e_n) \neq \emptyset$; and
- since A_S is deterministic, $\mathcal{R}_{A_S}(e_0, \ldots, e_n)$ has at most one element.

Since A_P is historical, there is a single initial state p^0, and by the definition of μ, $\mu(p^0) = s^0$, where s^0 is the single initial state of A_S; thus (REμ1) is satisfied.

Finally to see that (REμ2) holds, let p, e, and p' be such that $p \xrightarrow{e}_P p'$. Then p is reachable over a unique finite behavior $e_0, \ldots, e_n$. Thus p' is reachable over the finite behavior $e_0, \ldots, e_n, e$. It follows that $\mathcal{R}_{A_S}(e_0, \ldots, e_n) = \{s\}$ and $\mathcal{R}_{A_S}(e_0, \ldots, e_n, e) = \{s'\}$ with $s \xrightarrow{e}_S s'$. Thus $s = \mu(p) \xrightarrow{e}_S \mu(p') = s'$. □

4.3.2 Prophecy Relation and Measure

As discussed in Section 2.3, it is natural to overcome the limitations of refinement measures by letting the progress measure μ express a *prophecy set*—a set of possible specification states. More specifically, if p is a program state reachable by $e_0, \ldots, e_n$, then $\mu(v)$ should be a set of specification states reachable by $e_0, \ldots, e_n$. Hence on a transition $p \xrightarrow{e}_P p'$, the value of the measure μ should change so that every $s' \in \mu(p')$ is reachable. This can be assured by requiring that to every state s' in $\mu(p')$ there corresponds a state s in $\mu(p)$ such that $s \xrightarrow{e}_S s'$. Thus we define:

Definition 2 The *prophecy relation* $\triangleright_{\mathrm{PR}}$ of a transition relation $\rightarrow$ on V is the transition relation on $\mathcal{P}V$ given as:

$(\triangleright_{\mathrm{PR}})$ $\quad S \overset{e}{\triangleright}_{\mathrm{PR}} S'$ if $\forall s' \in S' : \exists s \in S : \; s \xrightarrow{e} s'$

An infinite $\triangleright_{\mathrm{PR}}$-related sequence of non-empty finite sets gives rise to an infinite $\rightarrow$-related sequence of states:

Lemma 1 (Prophecy Relation Lemma) If $S_0 \overset{e_0}{\triangleright}_{\mathrm{PR}} S_1 \overset{e_1}{\triangleright}_{\mathrm{PR}} \cdots$ and $S_i \neq \emptyset$ is finite for all i, then there is a sequence $s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} \cdots$ with $s_i \in S_i$ for $i \geq 0$.

Proof Construct a forest as follows. Each node is of the form $s_0 \xrightarrow{e_0} \cdots \xrightarrow{e_{n-1}} s_n$ such that $s_i \in S_i$ for $i \leq n$ and $s_i \xrightarrow{e_i} s_{i+1}$ for $i < n$; in particular, the roots are elements of S_0. The edges are of the form

$$(s_0 \xrightarrow{e_0} \cdots \xrightarrow{e_{n-1}} s_n \,,\, s_0 \xrightarrow{e_0} \cdots \xrightarrow{e_n} s_{n+1}).$$

Since S_i is finite, the forest is a finite collection of finitely branching trees. The forest is infinite, because for all n, it follows from $S_0 \overset{e_0}{\triangleright}_{\mathrm{PR}} S_1 \overset{e_1}{\triangleright}_{\mathrm{PR}} \cdots$ and $S_i \neq 0$ that there are some $s_0, \ldots, s_n$ such that $s_0 \xrightarrow{e_0} \cdots \xrightarrow{e_{n-1}} s_n$ is a node. Hence by König's Lemma, there is an infinite path through one of the trees. This path defines $s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} \cdots$. □

A prophecy measure maps each program state to a finite set of specification states:

Definition 3 A *prophecy measure* μ for (A_P, A_S) is a mapping $\mu : V_P \to \mathcal{F}V_S$ such that:

(PRμ1) $p \in V_P^0 \Rightarrow \mu(p) \subseteq V_S^0$

(PRμ2) $p \xrightarrow{e}_P p' \Rightarrow \mu(p) \overset{e}{\triangleright}_{\mathrm{PR}} \mu(p')$

(PRμ3) $\mu(p) \neq \emptyset$

where $\triangleright_{\mathrm{PR}}$ is the prophecy relation of $\rightarrow_S$.

Prophecy measures give a sound and complete verification method for historical program automata and safety specification automata:

Proposition 3 Let A_P be an historical automaton (assumed complete according to the discussion in Section 4.1) and A_S a safety automaton. Then $L(A_P) \subseteq L(A_S)$ iff (A_P, A_S) has a prophecy measure.

"$\Leftarrow$" Assume that (A_P, A_S) has a prophecy measure μ. Let $p_0 \xrightarrow{e_0}_P p_1 \xrightarrow{e_1}_P \cdots$ be a run of A_P. By (PRμ2), $\mu(p_0) \overset{e_0}{\triangleright}_{\mathrm{PR}} \mu(p_1) \overset{e_1}{\triangleright}_{\mathrm{PR}} \cdots$, and by (PR$\mu$3), $\mu(p_i) \neq \emptyset$ for $i \geq 0$. We can use the Prophecy Relation Lemma to obtain a sequence $s_0 \xrightarrow{e_0}_S s_1 \xrightarrow{e_1}_S \cdots$, where $s_0 \in \mu(p_0)$. By (PRμ1), $s_0 \in \mu(p_0) \subseteq V_S^0$, whence $s_0 \xrightarrow{e_0}_S s_1 \xrightarrow{e_1}_S \cdots$ is a run of A_S. Note that the proof of this direction does not depend on any assumptions about A_P or A_S.

"$\Rightarrow$" Assume $L(A_P) \subseteq L(A_S)$. Define $\mu(p) = \mathcal{R}_{A_S}(e_0, \ldots, e_n)$, where $e_0, \ldots, e_n$ is the unique finite behavior such that $p \in \mathcal{R}_{A_P}(e_0, \ldots, e_n)$; by the assumption that A_P is complete, there is a sequence $e_0, \ldots, e_n$, and by the assumption that A_P is historical, this sequence is unique. By the assumption that A_S is a safety automaton, $\mu(p)$ is finite.

Since A_P is complete, there are $e_{n+1}, e_{n+2}, \ldots$ such that $e_0, e_1, \ldots \in L(A_P)$. Since $L(A_P) \subseteq L(A_S)$, $\mathcal{R}_{A_S}(e_0, \ldots, e_n) \neq \emptyset$. Hence for all p, $\mu(p)$ is nonempty, i.e. (PRμ3) holds.

By the assumption that A_S is historical, there is a single initial state p^0, and by the definition of μ, $\mu(p^0) = V_S^0$; thus (PRμ1) is satisfied.

Finally to prove that (PRμ2) holds, let p, e, and p' be such that $p \xrightarrow{e}_P p'$. There is a unique finite word $e_0, \ldots, e_n$ such that $p \in \mathcal{R}_{A_P}(e_0, \ldots, e_n)$. Define

$$S' = \{s' \mid \exists s \in \mu(p) : s \xrightarrow{e}_S s'\}.$$

It can be seen that $S' = \mathcal{R}_{A_S}(e_0, \ldots, e_n, e) = \mu(p')$. By definition of the prophecy relation, $\mu(p) \overset{e}{\triangleright}_{\mathrm{PR}} \mu(p')$, whence (PR$\mu$2) holds. □

It follows from the discussion in Section 2.3 that the method of prophecy measures is not complete if the restriction that A_P be historical is removed. This is because a program state p generally can be reached by many finite behaviors—each behavior giving rise to a different prophecy set. In the next section, we overcome this problem by instead imposing a further restriction on the specification automaton.

4.3.3 History Relation and Measure

Assume now that $L(A_P) \subseteq L(A_S)$ and that specification automaton A_S is deterministic. Consider a program state p. It can be reached by different finite behaviors $e_0, \ldots, e_n$. Let the measure $\mu(p)$ be the *history set*—the set of specification states that are reached by these finite behaviors (there is one such state per behavior because A_S is deterministic). On a transition $p \xrightarrow{e}_P p'$ and for each state $s \in \mu(p)$, there must be a state $s' \in \mu(p')$ such that $s \xrightarrow{e}_S s'$; this ensures that every partial run (history) of A_S can be extended. Thus we define:

Definition 4 The *history relation* $\triangleright_{\text{HI}}$ of a transition relation $\rightarrow$ on V is the transition relation on $\mathcal{P}V$ given as:

$(\triangleright_{\text{HI}})$ $\quad C \overset{e}{\triangleright}_{\text{HI}} C'$ if $\forall s \in C : \exists s' \in C' : \; s \xrightarrow{e} s'$

The history relation of $\rightarrow$ has the following property:

Lemma 2 (History Relation Lemma) If $C_0 \overset{e_0}{\triangleright}_{\text{HI}} C_1 \overset{e_1}{\triangleright}_{\text{HI}} \cdots$, then for all $s_0 \in C_0$, there is a sequence such that $s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} \cdots$ with $s_i \in C_i$ for all i.

Proof Let s_0 be any state in C_0. Then by definition of $\triangleright_{\text{HI}}$, there is a state s_1 in C_1 such that $s_0 \xrightarrow{e_0} s_1$. By iterating this argument, we obtain $s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} \cdots$ such that for all i, $s_i \in C_i$. □

An history measure maps a program state to a possibly infinite set of specification states:

Definition 5 A *history measure* μ for (A_P, A_S) is a mapping $\mu : V_P \to \mathcal{P}V_S$ such that

(HIμ1) $p \in V_P^0 \Rightarrow \exists s \in \mu(p) : s \in V_S^0$

(HIμ2) $p \xrightarrow{e}_P p' \Rightarrow \mu(p) \overset{e}{\triangleright}_{\mathrm{HI}} \mu(p')$

where $\triangleright_{\mathrm{HI}}$ is the history relation of $\rightarrow_S$.

History measures give a complete verification method for deterministic specification automata:

Proposition 4 Let A_P be an automaton (assumed complete according to the discussion in Section 4.1) and A_S a deterministic automaton. Then $L(A_P) \subseteq L(A_S)$ iff (A_P, A_S) has an history measure.

Proof "$\Leftarrow$" Let $p_0 \xrightarrow{e_0}_P p_1 \xrightarrow{e_1}_P \cdots$ be a run of A_P. By (HIμ1) there is an $s_0 \in \mu(p_0)$ such that $s_0 \in V_S^0$. By (HIμ2), $\mu(p_0) \overset{e_0}{\triangleright}_{\mathrm{HI}} \mu(p_1) \overset{e_1}{\triangleright}_{\mathrm{HI}} \cdots$. Thus by the History Relation Lemma, there is a run $s_0 \xrightarrow{e_0}_S s_1 \xrightarrow{e_1}_S \cdots$ of A_S.

"$\Rightarrow$" Assume $L(A_P) \subseteq L(A_S)$. Define $\mu(p) = \bigcup_{p \in \mathcal{R}_{A_P}(e_0,\ldots,e_n)} \mathcal{R}_{A_S}(e_0, \ldots, e_n)$, where the union is over all finite behaviors $e_0, \ldots, e_n$ such that $p \in \mathcal{R}_{A_P}(e_0, \ldots, e_n)$.

To prove (HIμ1), let $p \in V_P^0$. Note that $\mathcal{R}_{A_S}()$ contains the initial state s^0 of A_S because A_P is complete. Since $p \in \mathcal{R}_{A_P}()$, it follows that $\mathcal{R}_{A_S}() \subseteq \mu(p)$; thus $s^0 \in \mu(p^0)$.

To see that (HIμ2) holds, let p, e, p', and s be such that $p \xrightarrow{e}_P p'$ and $s \in \mu(p)$. Thus there is a sequence $e_0, \ldots, e_n$ such that $s \in \mathcal{R}_{A_S}(e_0, \ldots, e_n)$ and $p \in \mathcal{R}_{A_P}(e_0, \ldots, e_n)$. Since A_S is deterministic, $\mathcal{R}_{A_S}(u)$ is the singleton $\{s\}$. It follows that $\mathcal{R}_{A_S}(e_0, \ldots, e_n, e)$ is a set $\{s'\}$ such that $s \xrightarrow{e}_S s'$ and that $s' \in \mu(p')$, because $p' \in \mathcal{R}_{A_P}(e_0, \ldots, e_n, e)$. Thus (HI$\mu$2) holds. □

4.3.4 ND Relation and Measure

We have discussed relations that give complete methods for two special cases above: prophecy relations when A_P is historical and history relations when A_S is deterministic. Our solution to the general case consists of combining these relations: the ND relation is the history relation of the prophecy relation.

Definition 6 The *ND relation* $\triangleright_{\mathrm{ND}}$ on $\mathcal{PF}V$ of a transition relation $\rightarrow$ on V is defined as:

$$(\triangleright_{\mathrm{ND}}) \qquad C \overset{e}{\triangleright}_{\mathrm{ND}} C' \quad \text{if} \quad \forall S \in C : \exists S' \in C' : \; \forall s' \in S' : \exists s \in S : \; s \xrightarrow{e} s'$$

An immediate consequence of the two preceding lemmas is:

Lemma 3 (ND Relation Lemma) If $C_0 \overset{e_0}{\triangleright}_{\mathrm{ND}} C_1 \overset{e_1}{\triangleright}_{\mathrm{ND}} \cdots$, $S_0 \in C_0$, and $\emptyset \notin C_i$ for all i, then there is a sequence $s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} \cdots$ with $s_0 \in S_0$.

Proof As $S_0 \in C_0$ and as $C_0 \overset{e_0}{\triangleright}_{\mathrm{HI}} C_1 \overset{e_1}{\triangleright}_{\mathrm{HI}} \cdots$, there is by the History Relation Lemma a sequence $S_0 \overset{e_0}{\triangleright}_{\mathrm{PR}} S_1 \overset{e_1}{\triangleright}_{\mathrm{PR}} \cdots$ with $S_i \in C_i$.

Moreover since $\emptyset \notin C_i$, i.e. $S_i \neq \emptyset$, and since S_i is finite for all i, it follows by the Prophecy Relation Lemma that there is a sequence $s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} \cdots$ such that $s_0 \in S_0$. □

An ND measure μ associates to each program state a (history) set of (prophecy) sets of specification states:

Definition 7 An ND measure μ for (A_P, A_S) is a mapping $\mu : V_P \rightarrow \mathcal{PF}V_S$ such that

(NDμ1) $\quad p \in V_P^0 \;\Rightarrow\; \exists S \in \mu(p) : S \subseteq V_S^0$

(NDμ2) $\quad p \xrightarrow{e}_P p' \;\Rightarrow\; \mu(p) \overset{e}{\triangleright}_{\mathrm{ND}} \mu(p')$

(NDμ3) $\quad \emptyset \notin \mu(p)$

The main result of this chapter is:

Theorem 1 Let A_P be an automaton (assumed complete according to the discussion in Section 4.1) and A_S a safety automaton. Then, $L(A_P) \subseteq L(A_S)$ iff (A_P, A_S) has an ND measure.

Proof "⇐" Let $p_0 \xrightarrow{e_0}_P p_1 \xrightarrow{e_1}_P \cdots$ be a run of A_P. By (NDμ2), $\mu(p_0) \overset{e_0}{\triangleright}_{\mathrm{ND}} \mu(p_1) \overset{e_1}{\triangleright}_{\mathrm{ND}} \cdots$, and by (ND$\mu$1), there is a set $S_0 \in \mu(p_0)$ such that $S_0 \subseteq V_S^0$. Thus by (NDμ3) and the ND Relation Lemma, there is $s_0 \xrightarrow{e_0}_S s_1 \xrightarrow{e_1}_S \cdots$ such that $s_0 \in S_0 \subseteq V_S^0$. Therefore $s_0 \xrightarrow{e_0}_S s_1 \xrightarrow{e_1}_S \cdots$ is a run of A_S.

"⇒" Assume $L(A_P) \subseteq L(A_S)$. Define μ such that $S \in \mu(p)$ iff there is a finite behavior $e_0, \ldots, e_n$ such that $p \in \mathcal{R}_{A_P}(e_0, \ldots, e_n)$ and $S = \mathcal{R}_{A_S}(e_0, \ldots, e_n)$. By

the assumption that A_P is complete, there is for any such $e_0, \ldots, e_n$ a sequence $e_{n+1}, e_{n+2}, \ldots$ such that $e_0, e_1, \ldots \in L(A_P)$. Thus as $L(A_P) \subseteq L(A_S)$, it follows that $\mathcal{R}_{A_S}(e_0, \ldots, e_n) \neq \emptyset$. Hence for all p, $\mu(p)$ is a set of nonempty sets, i.e. in particular (NDμ3) holds.

To prove that μ satisfies (NDμ1), assume that $p \in V_P^0$. By the definition of μ, for all $p^0 \in V_P^0$, it holds that $V_S^0 \in \mu(p^0)$, whence (NDμ1) is satisfied.

To prove that (NDμ2) holds, let p, e, and p' be such that $p \xrightarrow{e}_P p'$ and let $S \in \mu(p)$. Thus there is a finite behavior $e_0, \ldots, e_n$ such that $p \in \mathcal{R}_{A_P}(e_0, \ldots, e_n)$ and $S = \mathcal{R}_{A_S}(e_0, \ldots, e_n)$. Define

$$S' = \{s' \mid \exists s \in S : s \xrightarrow{e}_S s'\}.$$

It can be seen that $S' = \mathcal{R}_{A_S}(e_0, \ldots, e_n, e) \in \mu(p')$. By definition of the prophecy relation, $S \overset{e}{\triangleright}_{\mathrm{PR}} S'$, whence (ND$\mu$2) holds. □

4.4 Derivation of Previous Methods

In this section we derive Abadi and Lamport's method [1] from our ND-measures. Along the way we obtain the method of Merritt [38] and Sistla [46].

Formulated in our terminology, the goal of [1] is to show $L(A_P) \subseteq L(A_S)$ by means of a refinement measure. To do this Abadi and Lamport add *history* and *prophecy information* to the program automaton before the refinement mapping is constructed. This information is such that one can verify locally that the language accepted does not shrink when it is added. The main result of [1] is that $L(A_P) \subseteq L(A_S)$ iff there is an automaton $A\hat{p}$—obtained by adding first history, then prophecy information to A_P—and a refinement measure of $(A\hat{p}, A_S)$. The papers [38,46] represent an intermediate approach between ours and that of [1], relying on modifying the program automaton and using a prophecy measure: $L(A_P) \subseteq L(A_S)$ iff there is an automaton $A_{P'}$—obtained by adding history information to A_P—and a prophecy measure of $(A_{P'}, A_S)$.

4.4.1 Adding History Information

Using ND measures, we first derive the method of [38,46].

Definition 8 We say that $A_{P'} = (\Sigma, V_{P'}, \rightarrow_{P'}, V^0_{P'})$ is obtained from A_P by *adding history information* if $V_{P'} \subseteq V_P \times I$—with I countable—and

(HI1) $\quad p \in V^0_P \;\Rightarrow\; \exists i : (p,i) \in V^0_{P'}$

(HI2) $\quad p \rightarrow_P p' \;\wedge\; (p,i) \in V_{P'} \;\Rightarrow\; \exists i' : (p,i) \xrightarrow{e}_{P'} (p',i')$

Note that (HI1) and (HI2) are equivalent to saying that μ defined by $\mu(p) = \{(p,i) \in V_{P'}\}$ is an history measure for $(A_P, A_{P'})$. Also observe that $A_{P'}$ is not necessarily complete or safety, even if A_P is.

Proposition 5 If $A_{P'}$ is obtained from A_P by adding history information, then $L(A_P) \subseteq L(A_{P'})$.

Proof As noted above, $\mu = \{(p,i) \in V_{P'}\}$ is an history measure for $(A_P, A_{P'})$. Thus by Proposition 4, $L(A_P) \subseteq L(A_{P'})$ holds. □

The method of [38,46] now follows from Theorem 1:

Corollary 1 (of Theorem 1) Let A_P be a complete automaton and let A_S be a safety automaton. Then $L(A_P) \subseteq L(A_S)$ iff there is an automaton $A_{P'}$—obtained by adding history information to A_P—and a prophecy measure for $(A_{P'}, A_S)$.

Proof "$\Leftarrow$" By Proposition 5, $L(A_P) \subseteq L(A_{P'})$, and by Proposition 3, $L(A_{P'}) \subseteq L(A_S)$.

"$\Rightarrow$" By Theorem 1 there is an ND measure μ_{ND} for (A_P, A_S). Let $I = \mathcal{F}V_S$, $V_{P'} = \{(p,S) \mid S \in \mu_{\mathrm{ND}}(p)\}$, $V^0_{P'} = \{(p,S) \in V_{P'} \mid p \in V^0_P, S \subseteq V^0_S\}$, and $(p,S) \xrightarrow{e}_{P'} (p',S')$ if $p \xrightarrow{e}_P p'$ and $S \overset{e}{\triangleright}_{\mathrm{PR}} S'$.

Now it can be seen that $A_{P'}$ is obtained from A_P by adding history information; in fact, (HI1) is satisfied, because if $p \in V^0_P$, then by (NDμ1) there is a $S \in \mu_{\mathrm{ND}}(p)$ such that $S \subseteq V^0_S$, thus $(p,S) \in V^0_{P'}$. Similarly it can be seen that (HI2) follows from (NDμ2).

To finish the proof, we define the prophecy measure for $(A_{P'}, A_S)$ by $\mu(p,S) = S$. Then (PRμ1), (PRμ2), and (PRμ3) can be seen to hold. □

The completeness proofs of earlier methods [1,38,46] rely on changing A_P to an infinite-state automaton by adding information that records the past history of states. In contrast our analysis yields:

Observation 1 If A_P and A_S are finite-state, then $A_{P'}$ can be chosen to be finite-state.

Proof In the proof of Corollary 1, the number of different history sets is finite when V_P is finite. □

In light of this observation, the concepts of history measure and history information are a bit misleading in that distinguishing among histories of the program automaton is not a cardinal point—rather what matters is to distinguish among prophecy sets of the specification automaton.

4.4.2 Adding Prophecy Information

Next, to obtain the method of [1], we define:

Definition 9 $A_{P'} = (\Sigma, V_{P'}, V^0_{P'}, \rightarrow_{P'})$ is obtained from A_P by *adding prophecy information* if $V_{P'} \subseteq V_P \times I$—with I countable—and

(PR1) $p \in V^0_P \ \wedge \ (p,i) \in V_{P'} \ \Rightarrow \ (p,i) \in V^0_{P'}$

(PR2) $p \xrightarrow{e}_P p' \ \wedge \ (p',i') \in V_{P'} \ \Rightarrow \ \exists i : (p,i) \xrightarrow{e}_{P'} (p',i')$

(PR3) $\emptyset \neq \{i \mid (p,i) \in V_{P'}\}$ is finite

It is not hard to see that requiring (PR1), (PR2), and (PR3) is equivalent to stating that μ defined as $\mu(p) = \{(p,i) \in V_{P'}\}$ is a prophecy measure of $(A_P, A_{P'})$. Also observe that $A_{P'}$ is not necessarily a safety automaton, nor is it necessarily complete, even if A_P has these properties.

Proposition 6 If $A_{P'}$ is a safety automaton obtained from A_P by adding prophecy information, then $L(A_P) \subseteq L(A_{P'})$.

Proof Follows from Proposition 3 and from the observation above that $\mu(p) = \{(p,i) \in V_{P'}\}$ is a prophecy measure of $(A_P, A_{P'})$. □

Now a version of Theorem 2 of [1] can be formulated:

Corollary 2 (of Theorem 1) Let A_P be an automaton (assumed complete according to the discussion in Section 4.1) and let A_S be a safety automaton. Then $L(A_P) \subseteq L(A_S)$ if there is a safety automaton $A_{P''}$—obtained by adding first history, then prophecy information to A_P—and a refinement measure for $(A_{P''}, A_S)$.

Proof "$\Leftarrow$" By Proposition 5 and Proposition 6, $L(A_P) \subseteq L(A_{P''})$. Moreover, it is easy to see that every run of $A_{P''}$ induces a run of A_S; thus $L(A_P) \subseteq L(A_{P''}) \subseteq L(A_S)$.

"$\Rightarrow$" Assume $L(A_P) \subseteq L(A_S)$. By Theorem 1 there is an ND measure μ of (A_P, A_S). Let $A_{P''} = (\Sigma, V_{P''}, \rightarrow_{P''}, V^0_{P''})$, where $V^0_{P''}, V_{P''} \subseteq V_P \times (V_S \times \mathcal{F}V_S)$ are given by:

$$\begin{aligned} V_{P''} &= \{(p,s,S) \mid s \in S \in \mu(p)\} \\ V^0_{P''} &= \{(p,s,S) \mid s \in S \in \mu(p) \ \wedge \ p \in V^0_P \ \wedge \ S \subseteq V^0_S\} \end{aligned}$$

and $(p,s,S) \xrightarrow{e}_{P''} (p',s',S')$ if $p \xrightarrow{e}_P p'$, $s \xrightarrow{e}_S s'$, and $S \overset{e}{\triangleright}_{\mathrm{PR}} S'$.

Then it is not hard to see that $A_{P''}$ is obtained by adding prophecy information to $A_{P'}$ from the proof of Corollary 1. Also, it can be seen that μ_{re} defined by $\mu_{\mathrm{re}}(p,s,S) = s$ is a refinement measure. □

4.5 Discussion

We have obtained a direct method of verification for nondeterministic automata specifications. It hinges on two assumptions: that the specification automaton has only bounded nondeterminism and that the program automaton is complete.

The restriction to bounded nondeterminism is also imposed in previous methods; in fact, as discussed in Chapter 8, there are recursion-theoretic arguments showing that there does not exist any reasonable method for automata with unbounded nondeterminism.

Our restriction to complete program automata is also rooted in the laws of recursion theory: just to determine if an effectively presented nondeterministic automaton defines the empty set—that it has no infinite runs—is Π^1_1-complete (there is an easy reduction from the Π^1_1-complete problem of determining whether an effectively represented tree has only finite paths). On the other hand, all the

methods described here involve a second order existential quantification; i.e. each method is of the form: $L(A_P) \subseteq L(A_S)$ iff there is a relation (representing a measure and auxiliary information) such that some first-order conditions hold. Thus the methods are essentially Σ_1^1 and therefore cannot possibly be used for the general problem $L(A_P) \subseteq L(A_S)$, where A_P is nondeterministic and A_S is a safety automaton.

One can remedy this situation by reformulating the verification problem: A_P *simulates* A_S if each finite and infinite behavior of A_P is a behavior of A_S. Paradoxically the problem of determining whether nondeterministic A_P simulates safety automaton A_S—something that looks stronger than $L(A) \subseteq L(S)$—is computationally much easier. In fact it can be shown that this problem is Π_1^0-complete.

Thus in order to avoid dealing with the rather strange concept of complete automata, it is not surprising that earlier papers [1,38,46] are concerned with methods for showing that A_P simulates A_S. Whether one considers only infinite behaviors or both finite and infinite behaviors, there is no substantial difference in how automata are related—except for the treatment of reachable program states that are not parts of any run: in the first case they have to be excluded from consideration; in the second case they matter.

All our results are applicable for showing that A_P simulates A_S; the only change is that a measure becomes a partial function, because there may be unreachable states of A_P. For example, Theorem 1 becomes:

Theorem 1′ Let A_P be a nondeterministic automaton and A_S a safety automaton (with $V_S^0 \neq \emptyset$). Then A_P simulates A_S iff (A_P, A_S) has a partial ND measure.

In the statement of Theorem 1′ we define

Definition 7′ A *partial ND measure* μ for (A_P, A_S) is a partial mapping $\mu : V_P \rightharpoonup \mathcal{P}\mathcal{F}V_S$ such that $V_P^0 \subseteq \mathrm{dom}\mu$ and for all $p, p' \in \mathrm{dom}\mu$:

(NDμ1) $p \in V_P^0 \;\Rightarrow\; \exists S \in \mu(p) : S \subseteq V_S^0$

(NDμ2) $p \xrightarrow{e}_P p' \;\Rightarrow\; \mu(p) \overset{e}{\triangleright}_{\mathrm{ND}} \mu(p')$

(NDμ3) $\emptyset \notin \mu(p)$

Here the reachable states are defined by domμ, which can be identified using traditional assertional techniques.

4.6 Historical Notes

To compare automata languages, Park presented simulation relations, which we here call history measures, in [40]. History measures have appeared many times despite their incompleteness. For example, they were used in [48] for automata that involve temporal logic; in [34] for hierarchical proofs of program correctness; and in [24] for modeling distributed computing.

The approach of Abadi and Lamport [1] gives a verification method that essentially applies to the same kinds of problems considered here. In addition they show how safety and liveness issues can be separated by using automata that are equipped with auxiliary liveness properties. Their method is based on transforming the program automaton as explained in Section 4.4. The terminology "history" and "prophecy" used in this chapter is from [1]. Abadi and Lamport use stuttering automata, although this is not essential to their results. A stuttering automaton is one in which repetition of events is considered a single event. Abadi and Lamport have pointed out [2] that the method in [1] and our method, which was developed independently [27], are closely related. In fact they indicated that there are translations in both directions (we gave only the translation from ND measure to refinement measure).

The method of Sistla and Merritt [38,46] is an intermediate approach between that of [1] and ours. We showed in Section 4.4 that these methods are intimately related.

Chapter 5

Rabin Conditions

We turn to another issue of practical importance, namely liveness. Although intensively studied, many verification methods for liveness are rather indirect in that they depend on program transformations using recursive proof rules or auxiliary variables.

In this chapter we provide a concise mathematical characterization of what it means for a program to satisfy a certain kind of liveness property. To do this as simply as possible, we formulate the verification problem as a graph-theoretic problem. A graph represents computations, and a liveness property is expressed as a *Rabin condition* on the graph. We introduce *Rabin measures* on graphs for measuring progress towards satisfaction of a Rabin condition. The main result of this chapter is that a graph satisfies a finite Rabin condition iff it has a Rabin measure. A simple verification method for fair termination arises from this result; examples are given in Chapter 9.

The chapter is organized as follows. In Section 5.1 we define Rabin conditions on graphs. In Section 5.2 we define *pointer trees* and prove some basic lemmas about infinite sequences of nodes in trees. In Section 5.3 we introduce Rabin measures and state the main result, whose proof involves a rather involved transfinite construction given in Section 5.4. We apply the main result to verification with Rabin automata in Section 5.5. In Section 5.6 we briefly discuss the history of verification with Rabin conditions.

5.1 Rabin Conditions

Let V be a countable set of states. A *Rabin pair* (R, I) on V consists of a set $R \subseteq V$ of *reconfirming* states and a set $I \subseteq V$ of *invalidating* states. We say that an infinite sequence $v_0, v_1, \ldots$ of states *satisfies* (R, I) and write $v_0, v_1, \ldots \models (R, I)$ if there are infinitely many $v_k \in R$ and only finitely many $v_k \in I$.

A *Rabin condition* (or *Rabin assignment*) $\mathcal{C}$ is a set $\{(R_\chi, I_\chi) \mid \chi \in X\}$ of Rabin pairs. Here X is a countable set of *colors*, and Rabin pair (R_χ, I_χ) is said to have color χ. The set R_χ is the set of χ-reconfirming states and I_χ is the set of χ-invalidating states. For technical reasons, we always assume without loss of generality that $0 \in X$ and that $I_0 = \emptyset$ (one can always add the pair $(\emptyset, \emptyset)$ without changing the semantics of satisfaction defined next). We say that an infinite sequence $v_0, v_1, \ldots$ *satisfies* $\mathcal{C}$ and write $v_0, v_1, \ldots \models \mathcal{C}$ if for some χ, $v_0, v_1, \ldots \models (R_\chi, I_\chi)$. We say that a graph $G = (V, E)$ satisfies a Rabin condition $\mathcal{C}$ on V and write $G \models \mathcal{C}$ if every infinite path $v_0, v_1, \ldots$ in G satisfies $\mathcal{C}$.

5.2 Pointer Trees

Let *Ptr* be a (perhaps uncountable) set of *pointer values* with a total well-founded relation $\trianglerighteq$, called a *pointer relation*. We write $d \triangleright d'$ if $d \trianglerighteq d'$ and $d \neq d'$. A *pointer tree* T over *Ptr* is a prefix-closed countable subset of Ptr^*.[1] Each sequence $t = \langle t^1, \ldots, t^\ell \rangle$ in T represents a *node*, which has *children* $t \cdot \langle d \rangle \in T$. Here $d \in Ptr$ is the *pointer* to $t \cdot \langle d \rangle$ from t. If t' is a prefix of $t \in T$, then t' is called an *ancestor* of t. If $t' = t \cdot \langle \theta \rangle \in T$ then t is a *parent* of t'. We visualize pointer trees as growing upwards; see Figure 5.1, where $Ptr = \{0, 1, 2\}$ and children are depicted from left to right in descending order. Any sequence of pointers $t^1, t^2, \ldots$ (finite or infinite) denotes a *path*— $\langle\rangle, \langle t^1 \rangle, \langle t^1, t^2 \rangle, \ldots$ (finite or infinite) in T, provided each $\langle t^1, \ldots, t^\ell \rangle \in T$. The *level* $|t|$ of a node $t = \langle t^1, \ldots, t^\ell \rangle$ is the number ℓ; the level of $\langle\rangle$ is 0. The component t^ℓ of $t = \langle t^1, \ldots, t^n \rangle$, where $\ell \leq n$, is denoted $t[\ell]$ and called the *ℓ-pointer* of t. The *prefix up to level* λ of $t = \langle t^1, \ldots, t^n \rangle$ is $\langle t^1, \ldots, t^{\max\{n,\lambda\}} \rangle$, denoted $t{\downarrow}\lambda$. The *height* of T is the maximum node level (if it exists). T is *finite-path* if there are no infinite paths in T. A *common ancestor*

[1] In this thesis we consider only pointer trees equipped with a pointer relation. However, there exist progress measures that do not depend on such a relation.

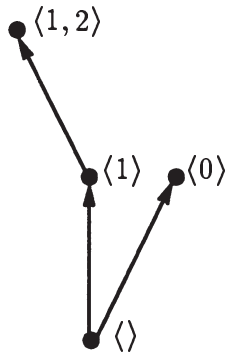


Figure 5.1: A pointer tree.

of nodes t and t' is a node $\tilde{t}$ that is a prefix of both of t and of t'. The common ancestor at the highest level is the *least common ancestor*.

A simple but central lemma about the infinite sequences of nodes in a pointer tree is:

Lemma 4 (Least Common Ancestor Lemma) Let $t_0, t_1, \ldots$ be any infinite sequence of nodes in a finite-path pointer tree T. Then there is a node t that is almost always a common ancestor and is infinitely often the least common ancestor of t_k, t_{k+1}.

Proof The root $\langle\rangle$ is always a common ancestor of t_k, t_{k+1}. Hence, if $\langle\rangle$ infinitely often is the least common ancestor, there is nothing more to prove. Otherwise for some number K^1, $\langle\rangle$ is not the least common ancestor of t_k, t_{k+1} for all $k \geq K^1$. It follows that $t_k[1] = t_{k+1}[1] = \cdots$; denote this value by d^1. Then $\langle d^1\rangle$ is a common ancestor of t_k, t_{k+1} for $k \geq K^1$. If $\langle d^1\rangle$ is a least common ancestor infinitely often, then there is nothing more to prove. Otherwise there is a number $K^2 \geq K^1$ such that $\langle d^1\rangle$ is a common ancestor, but not the least, of t_k, t_{k+1} for $k \geq K^2$.

Continuing this argument, either we obtain a node $\langle d^1, \ldots, d^\ell\rangle$ that is almost always a common ancestor and is infinitely often the least common ancestor, or we obtain an infinite path $d^1, d^2, \ldots$ in T. But the latter is not possible by the assumption that T is finite-path.

Note that this proof does not depend on the pointer relation of T. □

Definition 10 ($\succeq$ Ordering) The ordering $\succeq$ on T is defined by: $t \succeq t'$ if there is a λ such that $t{\downarrow}\lambda = t'{\downarrow}\lambda$ and either $\lambda = |t| < |t'|$ or $\lambda < |t|, |t'|$ and $t[\lambda+1] \triangleright t'[\lambda+1]$.

In other words $t \succeq t'$ if t is an ancestor of t' or if t' branches off to the right of t (assuming T is depicted as in Figure 5.1). This is a total order on T.

Definition 11 (Kleene-Brouwer Ordering) The ordering $\succ$ defined as $t \succ t'$ if $t \succeq t'$ and $t \neq t'$ is the *Kleene-Brouwer ordering*.

It is well known that the Kleene-Brouwer ordering is well-ordered [43]:

Lemma 5 (Kleene-Brouwer Ordering Lemma) If T is finite-path, then $\succ$ is well-ordered.

Proof Suppose for a contradiction that $t_0 \succ t_1 \succ \cdots$ is an infinite sequence. By the Least Common Ancestor Lemma there is a node t and number K such that for $k \geq K$, t is a common ancestor of t_k, t_{k+1} and is infinitely often the least common ancestor of t_k, t_{k+1}. Let $\lambda = |t|$ and let $k \geq K$. Whether t is the least common ancestor or just an ancestor of t_k and t_{k+1}, it follows from $t_k \succ t_{k+1}$ that $t_{k+1}[\lambda+1]$ is defined. Therefore, for $k > K$, if t is a least common ancestor of t_k and t_{k+1}, then $t_k[\lambda+1] \triangleright t_{k+1}[\lambda+1]$. Otherwise, if t is a common ancestor, but not the least, then $t_k[\lambda+1] = t_{k+1}[\lambda+1]$. We conclude that $t_{K+1}[\lambda] \trianglerighteq t_{K+2}[\lambda] \trianglerighteq \cdots$ is an infinite sequence with infinitely many strict inequalities. Thus there is an infinite descending chain in *Ptr*, but this contradicts the assumption that $\trianglerighteq$ is well-founded. Therefore there is no sequence $t_0 \succ t_1 \succ \cdots$. □

Rabin measures are based on colored pointer trees, which are defined by:

Definition 12 A *colored pointer tree* (T, ξ) is a pointer tree T with a partial mapping $\xi : T \rightrightarrows X$, where X is a set of *colors*, assigning a color $\xi(t) \in X$ to each node t in $\mathrm{dom}\xi$.

5.3 Rabin Relation and Measure

We can now formulate the Rabin relation:

Definition 13 (Rabin Relation) For a colored pointer tree (T,ξ) and a Rabin condition $\mathcal{C} = \{(R_\chi, I_\chi) \mid \chi \in X\}$ on V, the *Rabin relation* $\triangleright_{\mathrm{R}} \subseteq T \times V \times T$ is defined as:

$t \overset{v}{\triangleright}_{\mathrm{R}} t'$ if

(R) $t \succ t'$ or $v \in R_{\xi(\hat{t})}$, where $\hat{t}$ is the least common ancestor of t and t'

(I) $v \notin I_{\xi(\tilde{t})}$ for each common ancestor $\tilde{t}$ of t and t'

Note that (I) implies that if $t \overset{v}{\triangleright}_{\mathrm{R}} t'$ then every common ancestor must be colored by ξ.

When T is finite-path, the Rabin relation ensures in the limit that $\mathcal{C}$ is satisfied:

Lemma 6 (Rabin Relation Lemma) Let T be finite-path. For any sequence $t_0 \overset{v_1}{\triangleright}_{\mathrm{R}} t_1 \overset{v_2}{\triangleright}_{\mathrm{R}} \cdots$, it holds that $v_0, v_1, \ldots \models \mathcal{C}$.

Proof By the Least Common Ancestor Lemma, there is a node t that is (a) almost always a common ancestor and (b) infinitely often the least common ancestor of t_k, t_{k+1}. Let $\chi = \xi(t)$ and let $\lambda = |t|$. We show that $v_0, v_1, \ldots \models (R_\chi, I_\chi)$.

First, by (a) and (I) it holds almost always that $v_k \notin I_\chi$.

Second, it holds infinitely often that $v_k \in R_\chi$; for suppose that the contrary is true, i.e. that for some K and for all $k \geq K$, $v_k \notin R_\chi$. Then we can by (a) assume that K is such that for all $k \geq K$, t is a common ancestor of t_k and t_{k+1}. Now let $k \geq K$. If t is the least common ancestor of t_k and t_{k+1}, then $t_k \succ t_{k+1}$ by (R) and our supposition that $v_k \notin R_\chi$; in fact, $t_k{\downarrow}\,\lambda{+}1 \succ t_{k+1}{\downarrow}\,\lambda{+}1$. Otherwise, t is a common ancestor, but the not least, of t_k and t_{k+1}; thus $|t_k|, |t_{k+1}| > |t|$ and $t_k{\downarrow}\,\lambda{+}1 = t_{k+1}{\downarrow}\,\lambda{+}1$. We conclude that $t_K{\downarrow}\,\lambda{+}1 \succeq t_{K+1}{\downarrow}\,\lambda{+}1 \succeq \cdots$ and that due to (b) infinitely many inequalities are strict. This contradicts the Kleene-Brouwer Ordering Lemma. Thus it holds infinitely often that $v_k \in R_\chi$. □

Definition 14 A *Rabin measure* $(\mu, (T,\xi))$ of $(G,\mathcal{C})$, where $G = (V,E)$ and $\mathcal{C} = \{(R_\chi, I_\chi) \mid \chi \in X\}$, is a colored finite-path pointer tree (T,ξ) and a mapping $\mu : V \to T$ respecting the edge relation; i.e. $(u,v) \in E$ implies $\mu(u) \overset{v}{\triangleright}_{\mathrm{R}} \mu(v)$.

The main result of this chapter is:

Theorem 2 Let $\mathcal{C} = \{(R_\chi, I_\chi) \mid \chi \in X\}$ be a finite Rabin condition, i.e. $|X| < \omega$. Then $G \models \mathcal{C}$ iff $(G, \mathcal{C})$ has a Rabin measure (of height $\leq |X|$).

Proof ($\Leftarrow$) Let $v_0, v_1, \ldots$ be an infinite path in G. Then $\mu(v_0) \overset{v_1}{\underset{R}{\rhd}} \mu(v_1) \overset{v_2}{\underset{R}{\rhd}} \cdots$ Thus by Lemma 6, $v_0, v_1, \ldots \models \mathcal{C}$.

($\Rightarrow$) This is the subject of Section 5.4. □

5.4 Construction of Rabin Measures

We show how to construct a Rabin measure of $(G, \mathcal{C})$, where $G = (V, E)$ is a graph that satisfies a finite Rabin condition $\mathcal{C} = \{(R_\chi, I_\chi) \mid \chi \in X\}$.

5.4.1 Colorings of Graphs

We need some definitions before stating a key lemma. A *color set assignment* CS is a map $V \to \mathcal{P}X$. It associates a set of *permissible* colors $CS(v) \subseteq X$ to each vertex $v \in V$. An infinite path $v_0, v_1, \ldots$ is *eventually* χ*-permissible* if there is a K such that for all $k \geq K$, $\chi \in CS(v_k)$. A color set assignment is *eventually permissible* if every infinite path is eventually χ-permissible for some χ. A set $W \subseteq V$ is χ*-colorable* if for all $v \in W$, $\chi \in CS(v)$; in that case, χ is called a *valid* color of W. A set is *mono-colorable* if it is χ-colorable for some χ.

A *coloring* c of a set V is a mapping $c : V \to X \cup \{\bot\}$. If $c(v) = \bot$ then c does not assign a color to v, otherwise $c(v)$ is the color of v. A coloring c *obeys* a color set assignment CS if for all $v \in V$, $c(v) = \bot$ or $c(v) \in CS(v)$. An infinite path $v_0, v_1, \ldots$ is *eventually* χ*-stable* with respect to c, where $\chi \in X$, if there is a K such that for all $i \geq K$, $\chi = c(v_i)$. A coloring c is *eventually stable* if every infinite path is eventually χ-stable for some χ. A set $W \subseteq V$ is *monochromatic* with respect to c if there is $\chi \in X$ such that for all $v \in W$, $c(v) = \chi$. A *singular set* of V consists of a single loop-free vertex v such that $c(v) = \bot$; a vertex v is *loop-free* if $(v, v) \notin E$. The following graph-theoretic lemma is essential to the completeness proof.

Lemma 7 Let $G = (V, E)$ be a countable graph. If CS is an eventually permissible color set assignment, then there is an eventually stable partial coloring $c : V \to X \cup \{\perp\}$ obeying CS and a partition $\mathcal{S} = \{W_\theta\}_{\theta<\gamma}$ of V, where γ is a countable ordinal, such that

- each W_θ is either monochromatic or singular; and
- if $(v, v') \in E$, $v \in W_\theta$, and $v' \in W_{\theta'}$, then $\theta \geq \theta'$.

Proof We define an operator Γ, which will be applied transfinitely to simultaneously yield a coloring c and a partition $\mathcal{S}$ of V. Before doing this, we need some notation. For $\mathcal{S} \subseteq \mathcal{P}V$, i.e. for $\mathcal{S}$ a class of subsets of V, $\bigcup\mathcal{S} = \{v \mid \exists S \in \mathcal{S} : v \in S\}$. The set $\mathcal{R}(v, W)$, where $v \in W \subseteq V$, is $\{v' \mid v \to_W^* v'\}$, the set of vertices in W *reachable* from v by a path in W. The graph $G\backslash W$, for $W \subseteq V$, is defined as $G|(V\backslash W)$, the induced subgraph on $V\backslash W$.

Now let Γ be an operator that applied to a class $\mathcal{S} \subseteq \mathcal{P}V$ and a partial coloring c extends $\mathcal{S}$ and c according to (a) or (b) below. To make the definition of Γ unambiguous, we assume that in (a) there is a fixed enumeration of vertices v and that the least vertex fulfilling the condition is taken; similarly in (b), we assume a fixed enumeration of pairs (v, χ).

(a) Add to $\mathcal{S}$ the set $\{v\}$, if $v \in G\backslash\bigcup\mathcal{S}$ has no successors in $G\backslash\bigcup\mathcal{S}$. Moreover, define $c(v) = \perp$. Note that $\{v\}$ is then a singular set, since v has no self-loop.

(b) Add to $\mathcal{S}$ the set $\mathcal{R}(v, V\backslash\bigcup\mathcal{S})$, if (a) is not applicable and v, χ are such that $\mathcal{R}(v, V\backslash\bigcup\mathcal{S})$ is χ-colorable. Define $c(v') = \chi$ for all $v' \in \mathcal{R}(v, V\backslash\bigcup\mathcal{S})$. Note that $\mathcal{R}(v, V\backslash\bigcup\mathcal{S})$ becomes a monochromatic set.

The operator Γ so defined is obviously a monotone operator with respect to $\mathcal{S}$ and c. Let now $\mathcal{S}$ and c be the result of applying Γ transfinitely. It is clear that $\mathcal{S}$ is a class of disjoint subsets of V that are either singular or monochromatic. In fact, $\mathcal{S}$ is a partition of V:

Claim 1 $\bigcup\mathcal{S} = V$.

Proof Let $V_\infty = V\backslash\bigcup\mathcal{S}$ and assume that V_∞ is not empty. Then for every vertex $v_0 \in V_\infty$, there is an infinite path in $G|V_\infty$ originating in v_0. In fact, v_0 must have a successor v_1 in $G|V_\infty$,[2] because otherwise (a) would have been applied to remove

[2] A *successor* of a vertex u is a vertex v such that $(u, v) \in E$.

v_0 from V_∞. By iterating this argument, we conclude there is an infinite path $v_0, v_1, \ldots$ in V_∞.

Now, to prove that V_∞ is empty, we suppose for a contradiction that u_0 is a vertex in V_∞. Moreover we rename the colors in X so that $X = \{0, 1, \ldots, N-1\}$. The following procedure yields a sequence of vertices $u_0, v_1, u_1, v_1, \ldots$ in V_∞.

If $\mathcal{R}(u_0, V_\infty)$ is not 0-colorable, then there is some v_0 reachable in $G|V_\infty$ from u_0 such that $0 \notin CS(v_0)$ (perhaps v_0=u_0). Let u_1 be a successor of v_0 (possibly v_0 itself). If $\mathcal{R}(u_1, V_\infty)$ is not 1-colorable, then there is some vertex v_1 reachable from u_1 such that $1 \notin CS(v_1)$ (perhaps v_1=u_1).

Continuing in this way, we obtain either

- a vertex u_k such that $\mathcal{R}(u_k, V_\infty)$ is $(k \bmod N)$-colorable, or
- an infinite path containing the vertices $u_0, v_0, u_1, v_1, \ldots$ such that each u_k is not $(k \bmod N)$-colorable.

The second case would contradict the assumption that CS is eventually permissible. Therefore, there must be a $v \in V_\infty$ such that $\mathcal{R}(v, V_\infty)$ is mono-colorable, i.e. (b) can be applied. But this contradicts that $\mathcal{S}$ is the value obtained by transfinite applications of (a) and (b). Hence V_∞ is empty. □

To finish the proof, we need the following definition. For a class $W \in \mathcal{S}$, let θ be the successor ordinal corresponding to the stage of the transfinite definition when W was included in $\mathcal{S}$; designate W by W_θ. It follows that we can enumerate the classes of $\mathcal{S}$ as $\{W_\theta \mid \theta \leq \eta, \theta \text{ is successor ordinal}\}$. Thus, for each $v \in V$, there is exactly one θ such that $v \in W_\theta$.

Claim 2 If $(v, v') \in E$, $v \in W_\theta$, and $v' \in W_{\theta'}$, then $\theta \geq \theta'$.

Proof By construction, $\bigcup_{\eta<\theta} W_\eta$ is closed under successors. □

Claim 3 The partial coloring c is eventually stable and obeys CS.

That c obeys CS follows from (a) and (b). Let now $v_0, v_1, \ldots$ be an infinite path in V. Let $\theta(i)$ be the ordinal such that $v_i \in W_{\theta(i)}$. By Claim 2, $\theta(0) \geq \theta(1) \geq \cdots$. Thus, there is an ordinal θ and a number H such that for all $h \geq H$, $v_h \in W_\theta$. W_θ cannot be a singular class, because the vertex in such a class has no self-loop; thus

AssignRabin(t, Y, χ):

1. $\xi(t) := \chi$.
   $\widehat{W} := W(t) \backslash R_\chi$.
   $\hat{Y} := X \backslash (Y \cup \{\chi\})$.
2. Use Lemma 7 on $G|\widehat{W}$ with color set assignment $CS \cap \hat{Y}$ to obtain a partial coloring $c : \widehat{W} \to \hat{Y} \cup \{\perp\}$ obeying $CS \cap \hat{Y}$ and a partition $\mathcal{S} = \{W_\theta | \theta < \gamma\}$ of $\widehat{W}$ ($CS \cap \hat{Y}$ is explained in the text).
3. For each class W_θ of $\mathcal{S}$
   (a) $T := T \cup \{t \cdot \langle\theta\rangle\}$.
   (b) $W(t \cdot \langle\theta\rangle) := W_\theta$.
4. For each χ'-colored class W_θ of $\mathcal{S}$:
   AssignRabin$(t \cdot \langle\theta\rangle, Y \cup \{\chi\}, \chi')$.

Figure 5.2: Algorithm **AssignRabin**.

W_θ is monochromatic, and we conclude that $v_0, v_1, \ldots$ is eventually stable. □

Finally, if we renumber the partition $\mathcal{S} = \{W_\theta \,|\, \theta \leq \eta, \theta \text{ is successor ordinal}\}$, $\mathcal{S}$ can be written as $\{W_\theta \,|\, \theta < \gamma\}$, where γ is some ordinal (at most η). □

5.4.2 Proof of Theorem 2 "⇒"

Let $G = (V, E)$ be a graph that satisfies a finite Rabin condition $\mathcal{C}$. To construct a measure $(\mu, (T, \xi))$ of $(G, \mathcal{C})$, we use the algorithm **AssignRabin** in Figure 5.2. The algorithm builds the tree (T, ξ), whose pointers are ordinals, and labels each node $t \neq \langle\rangle$ of T with a set $W(t) \subseteq V$, which is to be the set of vertices mapped by μ to nodes having t as an ancestor. The purpose of **AssignRabin**(t, Y, χ) is to assign color χ to node t and to define the children $t \cdot \langle\theta\rangle$ of node t. Each child receives a label $W(t \cdot \langle\theta\rangle)$, which is a subset of $W(t)$. The set Y denotes the colors that have already been assigned to the proper ancestors of t.

Initially, **AssignRabin** is applied with parameters $(\langle\rangle, \emptyset, 0)$, and the tree T consists of only the root $\langle\rangle$ with label $W(\langle\rangle) = V$.

In Step 1 of **AssignRabin**, node t is assigned color χ, and $\widehat{W} = W(t) \backslash R_\chi$ is the set of states in $W(t)$ that are not χ-reconfirming. $\hat{Y} = X \backslash (Y \cup \{\chi\})$ is the set of colors that may be used to color the children of t.

In Step 2 Lemma 7 is used to obtain a partition $\mathcal{S} = \{W_\theta | \theta < \gamma\}$ of $\widehat{W}$ and a partial coloring c of $\widehat{W}$ obeying $CS \cap \hat{Y}$; here the fixed color set assignment CS is defined by $CS(v) = \{\chi \in X \mid v \notin I_\chi\}$, and $CS \cap \hat{Y}$ denotes the colors in CS that are also in $\hat{Y}$, i.e. $(CS \cap \hat{Y})(v) = CS(v) \cap \hat{Y}$.

In Step 3 a child $t \cdot \langle\theta\rangle$ is added to t for each class W_θ of $\mathcal{S}$ and $t \cdot \langle\theta\rangle$ is labelled W_θ.

Finally in Step 4 descendants of each child are constructed by further applications of **AssignRabin**.

To explain the construction of μ and to prove that Lemma 7 can indeed always be used in Step 2, we need some terminology.

We say that a subset $W \subseteq V$ is *χ-nonreconfirming* if $W \cap R_\chi = \emptyset$ and that W is *χ-noninvalidating* if $W \cap I_\chi = \emptyset$. A subset $W \subseteq V$ is *Y-nonreconfirming* if it is χ-nonreconfirming for each χ in Y. Similarly, a subset $W \subseteq V$ is *Y-noninvalidating* if it is χ-noninvalidating for each χ in Y.

Claim 4 For each application of the inductive procedure **AssignRabin**(t, Y, χ) starting with **AssignRabin**$(\langle\rangle, \emptyset, 0)$, the following hold:

- $|Y| = |t|$;
- $\chi \notin Y$;
- $W(t)$ is Y-nonreconfirming; and
- $W(t)$ is $(Y \cup \{\chi\})$-noninvalidating.

Proof (By induction) This is true for the first application **AssignRabin**$(\langle\rangle, \emptyset, 0)$ because $|Y| = |\emptyset| = |\langle\rangle| = |t|$; $\chi = 0$; and $W(t) = W(\langle\rangle) = V$, which is $\emptyset$-nonreconfirming and, since $I_0 = \emptyset$, $\{0\}$-noninvalidating.

When **AssignRabin**$(t \cdot \langle\theta\rangle, Y \cup \{\chi\}, \chi')$ is applied from within **AssignRabin** in Step 4, it may be assumed by the induction hypothesis that $|Y| = |t|$, $\chi \notin Y$, and $W(t)$ is $(Y \cup \{\chi\})$-noninvalidating and Y-nonreconfirming.

It follows that $|Y \cup \{\chi\}| = |t \cdot \langle\theta\rangle|$. Also, by definition of the color set assignment in Step 2, $\chi' \notin Y \cup \{\chi\}$. Furthermore, since W_θ is χ'-noninvalidating by this definition, W_θ is $(Y \cup \{\chi\} \cup \{\chi'\})$-noninvalidating. Finally, since W_θ is χ-nonreconfirming (because $W_\theta \subseteq \widehat{W} = W \backslash R_\chi$), it follows that $W(t \cdot \langle\theta\rangle) = W_\theta$ is $(Y \cup \{\chi\})$-nonreconfirming. □

Consider an application **AssignRabin**(t, Y, χ). By Claim 4, it follows that $\widehat{W}$ defined in Step 1 of **AssignRabin** is $(Y \cup \{\chi\})$-noninvalidating and $(Y \cup \{\chi\})$-nonreconfirming. Let now $v_0, v_1, \ldots$ be an infinite path in $\widehat{W}$. It is $(Y \cup \{\chi\})$-nonreconfirming because $\widehat{W}$ is $(Y \cup \{\chi\})$-nonreconfirming. Hence by the assumption that $G \models \mathcal{C}$, $v_0, v_1, \ldots \models (R_{\hat{\chi}}, I_{\hat{\chi}})$ for some $\hat{\chi} \in \hat{Y} = X \backslash (Y \cup \{\chi\})$. In particular, $v_0, v_1, \ldots$ is eventually $\hat{\chi}$-permissible with respect to the color set assignment $CS \cap \hat{Y}$ of $\widehat{W}$. Therefore, $CS \cap \hat{Y}$ is an eventually permissible color set assignment of $\widehat{W}$, and Lemma 7 is applicable in Step 2 of **AssignRabin**.

Now to show that all nodes constructed are at level $\leq N$, we note that if $|t| = |X| - 1$, then since $|Y| = |t|$ and $\chi \notin Y$ (by Claim 4), $Y \cup \{\chi\} = X$. In that case, the color set assignment of Step 2 assigns to each vertex the empty set; thus, the partition $\mathcal{S}$ contains only singular classes, and **AssignRabin** is not applied in Step 4. It follows that **AssignRabin** defines a tree T of height at most $|X|$. The tree has the following properties:

Claim 5

(a) For any $t \in T$ with $|t| < |X|$, $\{W(t \cdot \langle\theta\rangle) \mid t \cdot \langle\theta\rangle \in T\}$ is a partition of $W(t) \backslash R_{\xi(t)}$.

(b) For each $v \in V$ there is a unique maximal list $t = \langle t^1, \ldots, t^n \rangle$ with $1 \leq n \leq |X|$ such that $v \in W(t)$.

Proof (a) holds for $t = \langle\rangle$, because $W(\langle\rangle) = V$ and $\widehat{W}$ formed in **AssignRabin** is $W(\langle\rangle) \backslash R_{\xi(\langle\rangle)} = V \backslash R_0$; thus Lemma 7 in Step 2 and the definition of children in Step 4 yield a partition $\{W(\langle\theta\rangle) \mid \langle\theta\rangle \in T\}$ of $W(\langle\rangle) \backslash R_{\xi(\langle\rangle)}$. Similarly, by induction it can been seen that (a) holds for all $t \in T$ with $|t| < |X|$.

Part (b) follows from (a). □

Using Claim 5(b), we define $\mu : V \to T$ by $\mu(v) = \langle t^1, \ldots, t^n \rangle$. Now let $(u, v) \in E$ and let us prove that $\mu(u) \overset{v}{\underset{\mathrm{R}}{\triangleright}} \mu(v)$.

To prove (I), we note that if $\tilde{t}$ is a common ancestor of $\mu(u)$ and $\mu(v)$, then $\xi(\tilde{t})$ is defined, because otherwise $W(\tilde{t})$ would be a singular set containing both u and v—this is a contradiction because $(u, v) \in E$. By Claim 4 and by the definition of $\xi(\tilde{t})$ in Step 1, $W(\tilde{t})$ is $\xi(\tilde{t})$-noninvalidating. Since $v \in W(\tilde{t})$ by the definition of $\mu(v)$, it follows that $v \notin I_{\xi(\tilde{t})}$.

To prove (R), let $\hat{t}$ be the least common ancestor of $\mu(u)$ and $\mu(v)$. If $v \in R_{\xi(\hat{t})}$,

then (R) is trivially satisfied, so assume that $v \notin R_{\xi(\hat{t})}$. Then $|\mu(v)| > |\hat{t}|$ by Claim 5(a). Consider the application $\texttt{AssignRabin}(\hat{t}, Y, \xi(t))$.

If $u \in R_{\xi(\hat{t})}$, then $\mu(u) = \hat{t}$ because $u \notin \widehat{W} = W(t) \backslash R_{\xi(\hat{t})}$. Thus $\mu(u) = \hat{t} \succ \mu(v)$, because $|\mu(v)| > |\hat{t}|$. Otherwise, if $u \notin R_{\xi(\hat{t})}$, then by Claim 5(a), there are θ and θ' such that $u \in W_\theta$ and $v \in W_{\theta'}$. As $\hat{t} \cdot \theta$ is a prefix of $\mu(u)$, $\hat{t} \cdot \theta'$ is a prefix of $\mu(v)$, and $\hat{t}$ is the least common ancestor of $\mu(u)$ and $\mu(v)$, it follows that $\theta \neq \theta'$. Then by Lemma 7, $\theta > \theta'$, because $(u, v) \in E$. Therefore, $\mu(u) \succ \mu(v)$. Thus in all cases (R) holds. □

5.5 Automata-Theoretic Applications

In the literature of verification with automata, the focus is often on *conjunctive* acceptance conditions [6,37,45], because they are easy to handle when there is a method for each conjunct: simply apply that method for each of the conjuncts. Verification with disjunctive normal forms is not as obvious. In this section, however, we show that Rabin measures solve the problem of finding verification conditions for proving that a program satisfies a specification given by a deterministic Rabin automaton, whose acceptance condition is in a special disjunctive normal form. Combined with Safra's result [44], this yields a verification method for nondeterministic finite-state Büchi automata.

5.5.1 Verification with Deterministic Rabin Automata

A *deterministic Rabin automaton* $A = (\Sigma, V, \rightarrow, \{v^0\}, \mathcal{C})$ is a deterministic automaton $(\Sigma, V, \rightarrow, \{v^0\})$, as defined in Section 4.3, with a Rabin condition $\mathcal{C}$ on V. A run $v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots$ of A over a behavior $e_0, e_1, \ldots$ is *accepting* if $v_0, v_1, \ldots \models \mathcal{C}$. The language $L(A)$ *accepted by* A is the set of behaviors whose run is accepting.

We can now formulate verification conditions for showing that $L(A_P) \subseteq L(A_S)$, where the program automaton $A_P = (\Sigma, V_P, \rightarrow_P, V_P^0)$ is nondeterministic and the specification automaton $A_S = (\Sigma, V_S, \rightarrow_S, \{s^0\}, \mathcal{C})$ is a deterministic Rabin automaton. The verification conditions are formulated in terms of a *variant* $\mathcal{I}$, which relates the program and specification automata together with progress information for $\mathcal{C}$; formally, $\mathcal{I}$ is a relation on $V_P \times V_S \times T$, where (T, ξ) is a colored pointer tree of height $|X|$. The verification conditions for $\mathcal{I}$ are:

$(\mathrm{V1_R})$ $\quad p \in V_P^0 \;\Rightarrow\; \exists t : \mathcal{I}(p, s^0, t)$

$(\mathrm{V2_R})$ $\quad p \xrightarrow{e}_P p' \wedge \mathcal{I}(p,s,t) \;\Rightarrow\; \exists s', t' : s \xrightarrow{e}_S s' \wedge t \overset{s'}{\triangleright}_{\mathrm{R}} t' \wedge \mathcal{I}(p', s', t')$

For an automata-theoretic formulation of Theorem 2, we define a *dead-end* state of an automaton to be a reachable state that is not part of any run.[3]

Corollary 3 (of Theorem 2) Let A_P be an automaton with no dead-end states. Then $L(A_P) \subseteq L(A_S)$ iff there is a variant $\mathcal{I}$ satisfying $(V1_R)$ and $(V2_R)$.

Proof "$\Leftarrow$": Let $p_0 \xrightarrow{e_0}_P p_1 \xrightarrow{e_1} \cdots$ be a run of A_P. By $(\mathrm{V1_R})$ and $(\mathrm{V2_R})$, there are $s_0, s_1, \ldots$ and $t_0, t_1, \ldots$ such that $s_0 = s^0$, $s_0 \xrightarrow{e_0}_S s_1 \xrightarrow{e_1}_S \cdots$, and $t_0 \overset{s_1}{\triangleright}_{\mathrm{R}} t_1 \overset{s_2}{\triangleright}_{\mathrm{R}} \cdots$. It follows that $s_0 \xrightarrow{e_0}_S s_1 \xrightarrow{e_1}_S \cdots$ is a run of A_S, which is accepting by Lemma 6.

"$\Rightarrow$" Assume that $L(A_P) \subseteq L(A_S)$. Construct the joint graph $G = (V, E)$ defined by

$$V = \{(p,s) \mid p \in V_P \wedge s \in V_S \wedge \exists u \in \Sigma^* : (p \in \mathcal{R}_{A_P}(u) \wedge s \in \mathcal{R}_{A_S}(u))\}$$
$$E = \{((p,s),(p',s') \mid \exists e \in \Sigma : p \xrightarrow{e}_P p' \wedge s \xrightarrow{e}_S s'\}$$

where $\mathcal{R}_A$ is defined in Section 4.1.

Note that the assumption that A_P has no dead-end states implies that for any joint state (p,s), if there are e, p' such that $p \xrightarrow{e}_P p'$, then there is an s' such that $s \xrightarrow{e}_S s'$.

From the Rabin condition $\mathcal{C}$ on V_S, define a Rabin condition, also called $\mathcal{C}$, on G in the natural way. From the assumption $L(A_P) \subseteq L(A_S)$, it follows that $G \models \mathcal{C}$. Thus by Theorem 2 there is a Rabin measure $(\mu, (T, \xi))$ of G. Define $\mathcal{I}$ by

$$\mathcal{I}(p,s,t) \;\text{ iff }\; (p,s) \in V \wedge \mu(p,s) = t.$$

Then it is not hard to see that both $(\mathrm{V1_R})$ and $(\mathrm{V2_R})$ are fulfilled. □

5.5.2 Verification with Nondeterministic Büchi Automata

A Büchi automaton $A = (\Sigma, V, \rightarrow, V^0, V^F)$ is a nondeterministic automaton $(\Sigma, V, \rightarrow, V^0)$ with a set $V^F \subseteq V$ of *accepting states*. A run $v_0, v_1, \ldots$ of A over a

[3] A complete automaton has no dead-end states, but an automaton with no dead-end states is not necessarily complete as it may have non-reachable states that are not part of any run.

behavior $e_0, e_1, \ldots$ is *accepting* if there is a state in V^F that occurs infinitely often in $v_0, v_1, \ldots$ The language $L(A)$ *accepted by* A is the set of behaviors that allow an accepting run.

Using Safra's result [44], we define a method of verification for finite-state Büchi automata. Safra showed how to construct a deterministic Rabin automaton $\mathcal{R}(A_S)$ with $O(2^{n \log n})$ states and $O(n)$ pairs given a nondeterministic Büchi automaton A_S with n states. When applying our method of verification to $\mathcal{R}(A_S)$, we see that the variant becomes a relation $\mathcal{I}(p, r, t)$ where r is a state of $\mathcal{R}(A_S)$ and t is a node in a colored pointer tree of height n.

Corollary 4 (of Theorem 2) The verification conditions ($\mathrm{V1_R}$) and ($\mathrm{V2_R}$) applied to $\mathcal{R}(A_S)$ are sound and complete for verifying that $L(A_P) \subseteq L(A_S)$, where A_P is a nondeterministic automaton with no dead-end states and A_S is a finite-state Büchi automaton.

For further applications see [25], where the method for Rabin automata is extended to the ∀-automata of [37] and applied to simplify the method of [6]. Rabin measures apply to temporal logic conditions that are in the special form of a Rabin condition; progress measures for general disjunctive normal form formulas can be found in [26].

5.6 Historical Notes

For an excellent survey of the theory of automata on infinite strings, including the origins of Büchi and Rabin automata, see the chapter by Thomas in [31].

Alpern and Schneider [5,6] used deterministic finite-state *Büchi automata* as a method of specification. They obtained a method of verification for finite-state nondeterministic Büchi automata using the fact that any such automaton can be converted to a Boolean combination in conjunctive normal form of deterministic Büchi automata.[4] In [37] Manna and Pnueli gave verification conditions for ∀-*automata*, which are expressively equivalent to Büchi automata, although again there is no known easy conversion of a Büchi automaton into an ∀-automaton. Sistla [45] considered deterministic automata with acceptance conditions given as

[4]The probably simplest conversion is to apply Safra's determinization method and to write the Rabin acceptance condition in conjunctive normal form.

temporal formulas on automaton states with the modalities $F^{\infty}(f)$ (infinitely often f) and $G^{\infty}(f)$ (almost always f). He showed that sound and complete verification conditions exist for automata that are in a special conjunctive normal form in which each conjunct is a particularly simple disjunction.

Complete verification methods for termination under fairness were given in [18, 17,20,35,32,49]. These methods are based on *helpful directions* (identified in our method by the least common ancestor) and the iterative use of proof rules applied to syntactically transformed programs. Each program transformation corresponds to going up one level in our colored pointer trees. Thus behind the rather complicated syntactic program transformations of earlier methods lies the syntax-independent idea exposed here of measuring convergence based on colored trees.

The methods of *explicit schedulers* developed in [9,10,15] involve transforming programs by adding auxiliary variables that are nondeterministically assigned values determining fair computations. For an extensive treatment of fairness based on helpful directions and explicit schedulers, see [18].

A general approach to verifying with finite temporal formulas was proposed in [42]. It is based on establishing a direct correspondence between the program and the temporal formula, assigning a well-founded ordering to every program state. The verification conditions depend on inductively defined predicates on temporal formulas and are rather complicated.

For finite-state systems, many methods of verification have been devised, for example based on CTL* temporal logic [12,13,14], structural induction [29], and automata [51].

Chapter 6

Infinite Rabin Conditions

Let $\mathcal{C} = \{(R_\chi, I_\chi) \mid \chi \in X\}$ be an infinite Rabin condition. The Rabin Relation Lemma of the previous chapter shows that if there is a Rabin measure of $(G,\mathcal{C})$, then $G \models \mathcal{C}$. Yet it does not hold in general that $(G,\mathcal{C})$ has a Rabin measure when $G \models \mathcal{C}$. To see this, consider a vertex v of G through which infinitely many cycles pass. Assume moreover that for each cycle $C = v, v_1, \ldots, v_n, v, v_1, \ldots$ there is a Rabin pair $(R_{\chi(C)}, I_{\chi(C)})$ such that $C \models (R_{\chi(C)}, I_{\chi(C)})$; in addition, this Rabin pair must be the only one that C satisfies, and there must be infinitely many different $\chi(C)$. Now if there were a measure μ of $(G,\mathcal{C})$, then it would have to record in $\mu(v)$ the value $\chi(C)$ for all possible cycles C through v. This can clearly not be done using a Rabin measure, which only assigns a finite amount of information to each vertex.

Nevertheless, we show in this chapter that a Rabin measure exists when G is a tree and $G \models \mathcal{C}$. This result yields a practical verification method for extremely fair termination as explained in Chapter 10. We will study a mathematically more elegant verification technique for infinite Rabin conditions in Chapter 7.

6.1 Measures for Infinite Rabin Conditions

Theorem 3 Let $G = (V,E)$ be a tree and $\mathcal{C}$ an infinite Rabin condition on V. Then $G \models \mathcal{C}$ iff $(G,\mathcal{C})$ has a Rabin measure.

Proof "$\Leftarrow$" See Theorem 2.

"$\Rightarrow$" Assume that $G \models \mathcal{C}$ and that G has root v^0. We use the algorithm in Figure 6.1 to construct a Rabin measure $(\mu, (T, \xi))$. The algorithm defines for each

AssignInfRabin :

1. $u :=$ the least node in $V_{\text{do}} \backslash V_{\text{done}}$; if no such node exists, then exit.
   $n := |\mu(u)|$.
2. For each v such that $(u, v) \in E$:
   (a) $\kappa_I := \max\{\lambda \leq n \mid \forall \ell \leq \lambda : v \notin I_{c(u)[\ell]}\}$.
   (b) if there is an $\ell \leq \kappa_I$ such that $v \in R_{c(u)[\ell]}$ then
      i. $\kappa :=$ minimal such ℓ.
      ii. $\mu(v) := (\mu(u)\downarrow \kappa) \cdot \textbf{NewPointers}(\kappa, n+1)$.
   (c) else
      i. $\kappa := \kappa_I$.
      ii. $\mu(v) := (\mu(u)\downarrow \kappa) \cdot \textbf{NewPointers}(\kappa, n+1)$.
      iii. if $\kappa < n$ then $E[\kappa] := E[\kappa] \cup \{(\mu(u)[\kappa+1], \mu(v)[\kappa+1])\}$.
   (d) $c(v) := \begin{cases} c(u)[0..\kappa_I, \kappa_I+2..n, \kappa_I+1] \cdot \langle n+1 \rangle & \text{if } \kappa_I \leq n-2 \\ c(u)[0..n] \cdot \langle n+1 \rangle & \text{if } \kappa_I > n-2 \end{cases}$
   (e) $V_{\text{do}} := V_{\text{do}} \cup \{v\}$.
3. $V_{\text{done}} := V_{\text{done}} \cup \{u\}$.

where

NewPointers(ℓ', ℓ''):

1. For all ℓ, $\ell' \leq \ell \leq \ell''$:
   (a) $d_\ell :=$ some value not in $D[\ell]$.
   (b) $D[\ell] := D[\ell] \cup \{d_\ell\}$.
2. return $\langle d_{\ell'}, \ldots, d_{\ell''} \rangle$.

Figure 6.1: Algorithm **AssignInfRabin**.

vertex $v \neq v^0$ in G a node $\mu(v) = \langle t^1, \ldots, t^\lambda \rangle$ in T and a list $c(v) = \langle \chi^0, \ldots, \chi^\lambda \rangle$ defining the colors of the path $\mu(v)$ by $\xi(\mu(v)\downarrow \ell) = c(v)[\ell]$ (we use the notation from Section 5.2). Each application of **AssignInfRabin** *explores* a vertex u of V. **AssignInfRabin** uses variables V_{do}, the set of vertices marked for exploration, and V_{done}, the set of vertices that have been explored. For each $\ell \geq 1$, variable $D[\ell] \subset \omega$ is the subset of ω that is used to represent pointers at level ℓ, and $E[\ell] \subset \omega \times \omega$ represents a relation, denoted $>^\ell$, on $D[\ell]$.

The algorithm is initially applied with $V_{\text{do}} = \{v^0\}, V_{\text{done}} = \emptyset$, and for all ℓ, $D[\ell] = \emptyset$ and $E[\ell] = \emptyset$. Moreover, we assume that $\mu(v^0)$ and $c(v^0)$ are defined to

be $\langle\rangle$ and $\langle 0\rangle$; thus $\xi(\langle\rangle) = 0$. The algorithm works as follows.

In Step 1 the least vertex u marked for exploration that has not been explored is chosen; n denotes the length of the list $\mu(u)$, i.e. the level of node $\mu(u)$ in T.

In Step 2 vertex u is explored; this means that the values of $\mu(v)$ and $c(v)$ are calculated for all v such that $(u,v) \in E$. Step 2(a) defines the maximal level κ_I such that for all $\ell \leq \kappa_I$, v is not $c(u)[\ell]$-invalidating.

Next, κ—which will be the level of the least common ancestor of $\mu(u), \mu(v)$—is determined: if there is a level $\ell \leq \kappa_I$ such that v is $c(u)[\ell]$-reconfirming, then in 2(b) κ is defined as the minimal such level and $\mu(v)$ is defined to have a common prefix with $\mu(u)$ up to and including level κ, whereas for levels above, new pointers created by algorithm `NewPointers` are used. This defines the least common ancestor of $\mu(u), \mu(v)$ to be $\mu(u)\downarrow \kappa = \mu(v)\downarrow \kappa$. Note that μ is lengthened by one node, i.e. $|\mu(v)| = n+1 = |\mu(u)|+1$.

Otherwise, when there is no level $\ell \leq \kappa_I$ such that v is $c(u)[\ell]$-reconfirming, then in 2(c) κ is defined as κ_I; $\mu(v)$ is defined as before, but moreover $\mu(u)[\kappa+1] >^{\kappa} \mu(v)[\kappa+1]$ is enforced in Step 2(c)iii if $\kappa + 1 \leq n$.

In Step 2(d) the coloring of $\mu(v)$ is determined; here the notation $[\ldots]$ designates a permutation; for example, if $c = \langle 3,2,1\rangle$, then $c[3,1..2]$ is $\langle 1,3,2\rangle$. If $k_I \leq n-2$, then colors up to and including level κ_I are unchanged, whereas the color at level $\kappa_I + 1$ is moved to level n, and the colors above are moved down one level; in other words, the colors at levels $\kappa_I + 1$ to n are rotated in a downwards direction. Otherwise when $n-2 < \kappa_I \leq n$, all colors are left unchanged. Moreover, in both cases, the color $n+1$ is assigned to $\mu(v)\downarrow n+1$. This ensures that the colors in $c(v)$ are exactly $\{0,1,\ldots,|\mu(v)|\}$. Note that the color assigned at level 0 is always 0, because $c(v^0)[0] = 0$ and $c[v]$ remains 0, since for all v, it holds by the technical assumption from Section 5.1 that $v \notin I_0$; this implies that $\kappa > 0$, assuring the existence of κ_I in Step 2(a).

In Step 2(e) the vertex v is put in V_{do}, i.e. marked for exploration, and in Step 3 vertex u is put in V_{done}, since it has now been explored.

Claim 6 The algorithm `AssignInfRabin` defines a colored pointer tree (T,ξ) and a mapping $\mu : V \to T$ satisfying (R) and (I).

Proof From Step 1, Step 2(e), and Step 3 in Figure 6.1, it follows that for each application of `AssignInfRabin`, the sets V_{do} and V_{done} are augmented unless the

algorithm is terminated in Step 1 because $V_{\mathrm{do}} = V_{\mathrm{done}}$. Thus it can be seen that `AssignInfRabin` is a monotonic operator—with respect to the values of V_{do}, V_{done}, $D[\ell]$, and $E[\ell]$—and defines μ and c exactly once for each vertex $v \in V$.

Now let $t = \langle t^1, \ldots, t^n \rangle$ in T. The value of $\xi(t) = c(v)[n]$ is independent of the choice of $v \in V$ satisfying $\mu(v)\downarrow n = \langle t^1, \ldots, t^n \rangle$, because colors assigned in Step 2(d) are unchanged for levels less than κ, the level of the least common ancestor. It follows that c defines a unique coloring of T.

It is not hard to see that for $(u, v) \in E$, (R) and (I) are fulfilled. □

To complete the proof, we say that a node t of T is *active* in a sequence $t_0, t_1, \ldots$ if t is almost always a prefix of t_i and infinitely often the least common ancestor of t_i, t_{i+1}. The active node is unique if it exists.

Claim 7

(a) Let $v_0, v_1, \ldots$ be an infinite path in G. Then there is a node t that is active in $\mu(v_0), \mu(v_1), \ldots$ such that $v_0, v_1, \ldots \models (R_{\xi(t)}, I_{\xi(t)})$.

(b) For all λ, there is no infinite path $u_0, u_1, \ldots$ such that

$$\mu(u_0)[\lambda] \geq^{\lambda} \mu(u_1)[\lambda] \geq^{\lambda} \cdots$$

with infinitely many strict inequalities.

(c) For $d \in D^{\lambda}$, there is a vertex $\iota^{\lambda}(d) \in V$ such that for each vertex v in V with $\mu(v)[\lambda] = d$, $\iota^{\lambda}(d) \rightarrow^{*}_{\mu[\lambda]=d} v$; here, $\iota^{\lambda}(d) \rightarrow^{*}_{\mu[\lambda]=d} v$ means that v is reachable by a path $v_0, v_1, \ldots, v_n$ such that $v_0 = \iota^{\lambda}(d)$, $v_n = v$, and for all $i \leq n$, $\mu(v_i)[\lambda] = d$.

(d) If $d >^{\lambda} d'$, then there is a v such that $\iota^{\lambda}(d) \rightarrow^{*}_{\mu[\lambda]=d} v \rightarrow \iota^{\lambda}(d')$.

(e) $(D^{\lambda}, >^{\lambda})$ is well-founded.

(f) Let $t^{\ell} \in D^{\ell}$. Then there are unique $t^1, \ldots, t^{\ell-1}$ such that $\langle t^1, \ldots, t^{\ell} \rangle \in T$.

(g) T is finite-path.

Proof

(a) Let χ be a color such that $v_0, v_1 \ldots \models (R_{\chi}, I_{\chi})$. Such a color exists by the assumption that $G \models \mathcal{C}$. Then there is K such that for all $h \geq K$, $v_h \notin I_{\chi}$. Hence for $h \geq K$, the color χ can only move downwards in the stack. Thus

there is a level ℓ_χ and an $H \geq K$ such that for all $h \geq H$, $c(v_h)[\ell] = \chi$. Let $\hat{\chi}$ be the color such that $v_0, v_1 \ldots \models (R_{\hat{\chi}}, I_{\hat{\chi}})$ and $\ell_{\hat{\chi}}$ is minimal. Then κ is defined to be $\ell_{\hat{\chi}}$ infinitely often in Step 2(b), and κ takes on a lower value only finitely many times, due to the minimality of $\ell_{\hat{\chi}}$. Thus there is a node t colored $\hat{\chi}$ at level $\ell_{\hat{\chi}}$ that is active for $\mu(v_0), \mu(v_1), \ldots$ and such that $v_0, v_1, \ldots \models (R_{\xi(t)}, I_{\xi(t)})$.

(b) Suppose for a contradiction that $\mu(u_0)[\lambda]^\lambda \geq \mu(u_1)[\lambda] \geq^\lambda \cdots$ is such a path. Then κ of `AssignInfRabin` is almost always at least $\lambda - 1$, because when $\kappa < \lambda - 1$, the λ-pointers of $\mu(v_h), \mu(v_{h+1})$ are unrelated. Moreover, since infinitely many inequalities are strict, there is an active node t with $|t| = \lambda - 1$. Then for some number H and for each $h \geq H$, Step 2(b) was not taken; thus $v_h \not\models (R_{\xi(t)}, I_{\xi(t)})$. This contradicts (a).

(c) Among the vertices v such that $\mu(v)[\lambda] = d$, let $\iota^\lambda(d)$ be the first vertex to be included in V_{do}. It follows that every other vertex with this property is reachable from $\iota^\lambda(d)$, because a pointer is only unchanged from u to v when $(u, v) \in E$ (Step 2(a)ii and Step 2(b)ii).

(d) Assume $d >^\lambda d'$. Then there is an edge $(v, v') \in E$ such that $\mu(v)[\lambda] > \mu(v')[\lambda]$. By (c), $\iota^\lambda(d) \rightarrow^*_{\mu[\lambda]=d} v$. Also, $\iota^\lambda(d') = v'$ by (c) and the fact that G is a tree.

(e) Suppose for a contradiction that there is a λ and an infinite sequence $d_0 >^\lambda d_1 >^\lambda \cdots$. Then by (d) there are $v_0, v_1, \ldots$ such that

$$\iota^\lambda(d_0) \rightarrow^*_{\mu[\lambda]=d_0} v_0 \rightarrow \iota^\lambda(d_1) \rightarrow^*_{\mu[\lambda]=d_1} v_1 \rightarrow \cdots.$$

Thus there is an infinite path $u_0, u_1, \ldots$ in G satisfying (b); but this is a contradiction.

(f) This follows from the fact that any ℓ-pointer is preserved in Step 2(b)ii or Step 2(c)ii only if all lower pointers are preserved.

(g) Suppose for a contradiction that $\langle t^1, t^2, \ldots \rangle$ is an infinite path in T. For all $\ell < \ell'$, $\iota^\ell(t^\ell) \rightarrow^*_{\mu[\ell]=t^\ell} \iota^{\ell'}(t^{\ell'})$; in fact, use (f) to see that $\mu(\iota^{\ell'}(t^{\ell'}))\downarrow \ell' = \langle t^1, \ldots, t^\ell, \ldots, t^{\ell'} \rangle$, then use (c). Thus there is a path in G containing

$\iota^1(t_1), \iota^2(t_2), \ldots$ and there cannot be any active node for this path. This contradicts (a).

□

Thus T is a finite-path colored tree. It is not a pointer tree because at each level ℓ there is a different set D_ℓ of pointer values. This can be remedied by mapping the pointers into a countable ordinal θ that is the supremum of θ_ℓ, $\ell \geq 1$, where θ_ℓ is chosen so that D^ℓ can be mapped into θ_ℓ by an order homomorphism. □

6.2 Historical Notes

Infinite Rabin conditions correspond to the notion of an extreme fairness condition [41], which was studied as an approximation to probabilistic fairness. A method for extreme fairness based on changing the program by incorporating an explicit scheduler was proposed in [15].

Main [35] proved that the usual transformational proof rules for fairness [20, 32] are complete for extreme fairness.

Chapter 7

Liminf Specifications

We shall now take a more abstract approach to the verification problem. Although Rabin conditions are important for applications, they are rather cumbersome mathematical objects. So in order to present a pure theory of verification, we introduce in this chapter a simple but powerful way of specifying liveness properties. This yields a verification method which is applicable to all liveness properties that are $\mathbf{\Sigma_3^0}$ (i.e. countable unions of countable intersections of open sets) including extreme fairness.

We study graphs that are equipped with a *height assignment* that assigns a number to each vertex. We introduce the concepts of *Liminf condition* and *Liminf measure*, both expressed in terms of the graph and its height assignment. Our main result is that a graph satisfies a Liminf condition if and only if it has a Liminf measure. Applied to automata theory, this result gives a verification method for the class of deterministic *Liminf automata*, which are more expressive than Büchi automata. In descriptive set theory, we obtain a method based on finite approximations for comparing a $\mathbf{\Pi_3^0}$ program to a $\mathbf{\Sigma_3^0}$ specification.

Effective versions of our results state that an index for a measure can be obtained uniformly from indices of programs and specifications.

This chapter, which depends on Section 5.2 about pointer trees and orderings, is organized as follows. We introduce the notion of Liminf condition, relation, and measure in Section 7.1, and prove the main result. In Section 7.2 we provide an alternative method of measuring progress towards satisfaction of an infinite Rabin condition. In Section 7.3 we relate these concepts to descriptive set theory and give effective versions of our results. We give an automata-theoretic formulation of the

main result in Section 7.4. We show how to measure progress towards satisfaction of a negated Liminf condition in Section 7.5. We give brief historical notes in Section 7.6.

7.1 Liminf Condition, Relation, and Measure

Let $G = (V, E)$ be a countable directed graph whose infinite paths represent computations of a program. To specify the behavior of the program, we use a *height assignment* $h : V \to \omega$ that assigns a natural number to each vertex of G. Intuitively, an infinite path satisfies the specification given by h if the value of the height assignment dips down to some value infinitely often. Formally:

Definition 15 An infinite path $v_0, v_1, \ldots$ in G satisfies the *Liminf condition* with respect to h, written $v_0, v_1, \ldots \models \liminf h$, if $\liminf_{i\to\omega} h(v_i) < \omega$. Graph G satisfies the Liminf condition with respect to h, written $G \models \liminf h$, if every infinite path satisfies the Liminf condition.

Let t, t' be nodes in a pointer tree. The Liminf relation $\trianglerighteq_{\mathrm{Li}}$ is defined as:

Definition 16 (Liminf Relation) $t \trianglerighteq_{\mathrm{Li}} t'$ if either t is an ancestor of t', t' is an ancestor of t, or there is a level λ such that $|t|, |t'| \leq \lambda$, $t[\lambda] > t'[\lambda]$, and $t{\downarrow}\,\ell = t'{\downarrow}\,\ell$.

Intuitively, $t \trianglerighteq_{\mathrm{Li}} t'$ if either t' extends t or is a prefix of t or branches off to the right of t (assuming T is depicted as in Figure 5.1). Note that the Liminf relation is neither transitive nor antisymmetric. When T is finite-path, the Liminf relation guarantees that the liminf exists:

Lemma 8 (Liminf Relation Lemma) Let $t_0 \trianglerighteq_{\mathrm{Li}} t_1 \trianglerighteq_{\mathrm{Li}} \cdots$ be an infinite sequence of nodes in a finite-path pointer tree T. Then $\liminf_{i\to\omega} |t_i| < \omega$.

Proof By the Least Common Ancestor Lemma of Section 5.2, there is a node t such that t is almost always an ancestor of t_k and infinitely often the least common ancestor of t_k, t_{k+1}. We prove that infinitely often $t_k = t$. If not, then t is almost always a proper ancestor of t_k. Let $\lambda = |t|$. For sufficiently large k, if t is a least common ancestor of t_k and t_{k+1}, then $t_k[\lambda+1] > t_{k+1}[\lambda+1]$, because t is a proper ancestor of both t_k and t_{k+1}. Otherwise, if t is a common ancestor, but not the least, then $t_k[\lambda+1] = t_{k+1}[\lambda+1]$. We conclude that there is an infinite sequence $t_k[\lambda+1] \trianglerighteq t_{k+1}[\lambda+1] \trianglerighteq \cdots$ with infinitely many inequalities strict. This contradicts

the assumption that $\trianglerighteq$ is well-founded. Thus $\liminf_{i\to\omega} |t_i| = |t| < \omega$. □

To characterize graphs satisfying the Liminf condition, it is sufficient to study measures in pointer trees over ω.

Definition 17 A *Liminf measure* (μ, T) for (G, h) consists of a finite-path pointer tree $T \subset \omega^*$ and a mapping $\mu : G \to T$ such that

- for all v, $|\mu(v)| = h(v)$; and
- h respects the edge relation, i.e. if $(v, v') \in E$, then $\mu(v) \trianglerighteq_{\mathrm{Li}} \mu(v')$.

The main result of this chapter is:

Theorem 4 $G \models \liminf h$ if and only if (G, h) has a Liminf measure.

For the proof of this theorem, we need a lemma:

Lemma 9 Let $G = (V, E)$ be a nonempty countable graph. There exists a (possibly infinite) partition $\mathcal{S} = \{W_0, W_1, \ldots\}$ of V and a mapping ι that to each set W of $\mathcal{S}$ associates a node $\iota(W) \in W$ such that

- $\iota(W) \to_W^* W$; and
- for all $(v, v') \in E$, if $v \in W_k$ and $v' \in W_{k'}$, then $k \geq k'$.

Proof For $w \in W \subseteq V$, let $\mathcal{R}(w, W) = \{w' \mid w \to_W^* w'\}$. Assume that V is ordered $v_0, v_1, \ldots$ Let $W_0 = \{v | v \in \mathcal{R}(w_0, V)\}$, where w_0 is v_0, the least node in V. If $V \setminus \bigcup_{j<k} W_j$ is not empty, W_k is defined as $\mathcal{R}(w_k, V \setminus \bigcup_{j<k} W_j)$, where w_k is the least node in $V \setminus \bigcup_{j<k} W_j$. Then $W_0, W_1, \ldots$ are disjoint subsets of V. Either there is a k such that $\bigcup_{j<k} W_j = V$ or W_k is defined for all k. In the latter case, $\bigcup_j W_j = V$; otherwise, there is a least node v_h in $V \setminus \bigcup_j W_j$, but if k is the least number such that $v_0, v_1, \ldots, v_{h-1} \in \bigcup_{j<k} W_j$, then $v_h = w_k$ and $v_h \in W_k$. This is a contradiction.

Let $\iota(W_k) = w_k$. By construction, $\iota(W_k) \to_{W_k} W_k$. Suppose for a contradiction that $(v, v') \in E$ with $v \in W_k$ and $v' \in W_{k'}$ for $k < k'$. Then $v' \in \mathcal{R}(w_k, V \setminus \bigcup_j W_{j<k})$. Thus $v' \in W_k$, which is a contradiction. □

Proof (Theorem 4) "⇐" If $v_0, v_1, \ldots$ is an infinite path in G, we get $\liminf h(v_i) = \liminf |\mu(v_i)| < \omega$ by the Liminf Relation Lemma.

AssignLiminf(t):

1. $\widehat{W} := \{v \in W(t) \mid h(v) \geq |t|+1\}$.
   If $\widehat{W} = \emptyset$, then exit.
2. Use Lemma 9 to obtain a partition $\mathcal{S} = \{W_0, W_1, \ldots\}$ of $\widehat{W}$.
3. For each class W_k of $\mathcal{S}$:
   (a) $T := T \cup \{t \cdot \langle k \rangle\}$.
   (b) $W(t \cdot \langle k \rangle) := W_k$.
   (c) **AssignLiminf**($t \cdot \langle k \rangle$).

Figure 7.1: Algorithm **AssignLiminf**.

"$\Rightarrow$" Given that $G \models \liminf h$, we show how to construct a Liminf measure (μ, T) for (G, h). We use the algorithm **AssignLiminf** in Figure 7.1. In addition to T, it calculates a label $W(t) \subseteq V$ for each node $t \neq \langle\rangle$ in T; this label denotes the set of vertices that will be mapped by μ to a node having t as an ancestor. Consequently, $W(\langle\rangle)$ is defined to be V, because $\langle\rangle$ is an ancestor of every node in t. The transfinite algorithm **AssignLiminf** is initially applied at the root, i.e. $t = \langle\rangle$, and variable $T = \{\langle\rangle\}$.

The purpose of **AssignLiminf**(t) is to define the children $t \cdot \langle k \rangle$ of t together with their labels $W(t \cdot \langle k \rangle)$. First, nodes of height less than $|t|+1$ are filtered out. Second, the resulting set $\widehat{W}$ is partitioned using Lemma 9. Third, a child $t \cdot \langle k \rangle$ with label W_k is created for each class W_k of the partition, and the algorithm is reapplied on $t \cdot \langle k \rangle$.

The pointer tree T is the value of the variable T obtained by the transfinite applications of **AssignLiminf**(t) starting with $\langle\rangle$. Note that the construction of Lemma 9 applied in Step 2 associates to each label $W(t)$, $t \neq \langle\rangle$, a node $\iota(W(t))$ of G such that $\iota(W(t)) \rightarrow^*_{W(t)} W(t)$.

Claim 8

(1) For all $t \in T$, for all $v \in W(t)$, $h(v) \geq |t|$.

(2) For any $t \cdot \langle d \rangle \in T$, $W(t) \supseteq W(t \cdot \langle d \rangle)$. For each $v \in V$, there is a unique path $\langle d^1, \ldots, d^{h(v)} \rangle$ of T containing all nodes whose labels contain v.

Proof This follows by a straightforward induction. □

Now using (2) of Claim 8, we define $\mu(v) = \langle d^1, \ldots, d^{h(v)} \rangle$. We prove that μ is a Liminf measure.

First, it is obvious that $|\mu(v)| = h(v)$. Second, assume that $(u, v) \in E$. Let $\mu(u) = \langle d^1, \ldots, d^{h(u)} \rangle$, $\mu(v) = \langle e^1, \ldots, e^{h(v)} \rangle$ and $m = \min\{h(u), h(v)\}$. If $d^\ell = e^\ell$ for all $\ell \leq m$, then $\mu(u)$ is a prefix of $\mu(v)$ or $\mu(v)$ is a prefix of $\mu(u)$, thus $\mu(u) \trianglerighteq_{\mathrm{Li}} \mu(v)$. Otherwise, assume that $\lambda \leq m$ is such that $d^\lambda \neq e^\lambda$ and $d^\ell = e^\ell$ for all $\ell < \lambda$. Let $t = \langle d^1, \ldots, d^{\lambda-1} \rangle = \langle e^1, \ldots, e^{\lambda-1} \rangle$. Then `AssignLiminf`$(t)$ was applied and $u, v \in W(t)$. This invocation defined $W(t \cdot d^\lambda)$ and $W(t \cdot e^\lambda)$, which are disjoint subsets of $W(t)$ by the assumption that $d^\lambda \neq e^\lambda$. Hence, $d^\lambda > e^\lambda$ by Lemma 9. It follows that $\mu(u) \trianglerighteq_{\mathrm{Li}} \mu(v)$.

Claim 9 There is no infinite path in T.

Proof Suppose for a contradiction that $\langle d^1, d^2, \ldots \rangle$ is an infinite path. Then $W(\langle d^1 \rangle) \supseteq W(\langle d^1, d^1 \rangle) \supseteq \cdots$, thus for any $t = \langle d^1, \ldots, d^k \rangle \neq \langle\rangle$, $\iota(W(t)) \rightarrow^*_{W(t)} W(t \cdot \langle d^{k+1} \rangle)$, because $\iota(W(t)) \rightarrow^*_{W(t)} W(t)$ by Lemma 9. In particular, $\iota(W(t)) \rightarrow^*_{W(t)} \iota(W(t \cdot \langle d^{k+1} \rangle))$. We conclude that there exists an infinite path $v_0, v_1, \ldots$ in G and a monotonically increasing function $i(k)$ such that $v_{i(k)} = \iota(W(\langle d^1, \ldots, d^k \rangle))$ for $k \geq 0$, and for all $n \geq i(k)$, $h(v_n) \geq k$ by (1) of Claim 8. But then $v_0, v_1, \ldots$ does not satisfy the Liminf condition. This is a contradiction. □

7.2 Another Method for Rabin Conditions

We take a second look at the problem of verifying that a program satisfies an infinite Rabin condition. As in Chapter 6 we assume that a program is represented as a tree $G = (V, E)$ with root v^0. We show how to translate a Rabin condition $\mathcal{C}$ on V into a height assignment h.

Lemma 10 Let $\mathcal{C} = \{(R_i, I_i) \mid i \in \omega\}$ be a countable Rabin condition on a tree $G = (V, E)$. There is a function $f_{\mathcal{C}} : V \rightarrow \omega$ such that for any infinite path $v_0, v_1, \ldots$ in G:

$$v_0, v_1, \ldots \models \mathcal{C} \quad \text{iff} \quad \liminf_{k \rightarrow \omega} f_{\mathcal{C}}(v_k) < \omega.$$

Proof Define $f_i(u) = |u| - \mathbb{Q}R_i(u) + \mathbb{Q}I_i(u)$, where $|u|$ is the level of node u in the tree G and $\mathbb{Q}M(u)$, $M \in \{R_i, I_i\}$, is the level of the closest ancestor u' of u for which $u' \in M$:

$$\mathbb{Q}M(u) = \begin{cases} |u'| & \text{if } u' \text{ is the closest ancestor of } u \text{ such that } u' \in M \\ 0 & \text{if there is no such ancestor} \end{cases}$$

Then it is easy to see that $\liminf_{k\to\omega} f_i(v_k) < \omega$ iff $v_0, v_1, \ldots \models (R_i, I_i)$. Now let $f_C(u) = \min_{i<\omega} \rho(i, f_i(u))$, where $\rho : \omega \times \omega \to \omega$ is a bijection.

It remains to prove that $\liminf_{k\to\omega} f_C(v_k) < \omega$ iff there exists an i such that $\liminf_{k\to\omega} f_i(v_k) < \omega$.

If $\liminf_{k\to\omega} f_C(v_k) < \omega$, then there exists an h such that for infinitely many k, $f_C(v_k) = h$. Thus, there is an i such that $\rho(i, f_i(v_k)) = h$ for infinitely many k, whence $\liminf_{k\to\omega} f_i(v_k) < \omega$.

Conversely, if $\liminf_{k\to\omega} f_i(v_k) = j < \omega$ for some i and j, then for infinitely many k, $f_C(v_k) \leq \rho(i,j)$, hence $\liminf_{k\to\omega} f_C(v_k) \leq \rho(i,j) < \omega$. □

By Lemma 10 and Theorem 4, it follows that termination under extreme fairness can measured under assumption that the computation graph is a tree.

7.3 Applications to Descriptive Set Theory

The *Baire space* ω^ω is a topological space whose points consist of infinite sequences $\alpha = \alpha_0, \alpha_1, \ldots$ of natural numbers; see [39]. The *basic open sets* are $\{u \cdot \alpha | \alpha \in \omega^\omega\}$ for $u \in \omega^*$. The class of *open sets*, which are arbitrary unions of basic open sets, is denoted $\mathbf{\Sigma_1^0}$ (also called $\mathbf{G}$). The class of *closed sets*, which are the complements of open sets, is denoted $\mathbf{\Pi_1^0}$ (also called $\mathbf{F}$). (With this topology, a basic open set is also a closed set, and $\{u \cdot \alpha | \alpha \in \omega^\omega\}$, $u \in \omega^*$, constitute a *clopen basis*.)

The classes $\mathbf{\Sigma_1^0}$ and $\mathbf{\Pi_1^0}$ are at the first level of the *Borel hierarchy*. The class $\mathbf{\Sigma_2^0}$ (or $\mathbf{F}_\sigma$) consists of the countable unions of closed sets (whereas a countable intersection of $\mathbf{\Pi_1^0}$ sets is still a $\mathbf{\Pi_1^0}$ set). Similarly, the class $\mathbf{\Pi_2^0}$ (or $\mathbf{G}_\delta$) consists of the countable intersections of open sets. The class $\mathbf{\Sigma_3^0}$ (or $\mathbf{G}_{\delta\sigma}$) consists of the countable unions of $\mathbf{\Pi_2^0}$ sets, and the class $\mathbf{\Pi_3^0}$ (or $\mathbf{F}_{\sigma\delta}$) consists of the countable intersections of $\mathbf{\Sigma_2^0}$ sets.

Proceeding in this way, one obtains the *finite levels* $\mathbf{\Sigma_i^0}$ and $\mathbf{\Pi_i^0}$, $i \in \omega$, of the Borel hierarchy. The entire Borel hierarchy is the transfinite closure under countable unions and intersections of the basic open sets. The even larger class $\mathbf{\Sigma_1^1}$ of *analytic sets* consists of the continuous images of Borel sets. The class $\mathbf{\Pi_1^1}$ of *coanalytic sets*, which also include all Borel sets, consists of the complements of

analytic sets.

In this section we show how to determine $P \subseteq S$ from finite computations, where P is $\mathbf{\Pi}^0_3$ and S is $\mathbf{\Sigma}^0_3$.

7.3.1 Approximation Functions and Limit Operators

A limit operator is a way of defining certain sets in the Baire space as limits of finite sequences described by *approximation functions*. For example, let $f : \omega^* \to \{0,1\}$ be a binary approximation function and define the limit operator $\lim_{\mathbf{\Pi}^0_1}$ by:

$$\lim_{\mathbf{\Pi}^0_1} f = \{\alpha \in \omega^\omega \mid \forall u \propto \alpha : f(u) = 1\}$$

where $u \propto \alpha$ means that u is a finite prefix of α. Thus $\lim_{\mathbf{\Pi}^0_1} f$ is the set of infinite sequences α such that the value of f is 1 on all finite prefixes of α. It can be shown that any closed set is equal to $\lim_{\mathbf{\Pi}^0_1} f$ for some binary approximation function f; moreover for any f, $\lim_{\mathbf{\Pi}^0_1} f$ is a closed set. This characterization is essentially the same as the equivalence stated in Proposition 1 (Chapter 4) of closed sets and deterministic automata.

The classes $\mathbf{\Sigma}^0_1$, $\mathbf{\Pi}^0_2$, and $\mathbf{\Sigma}^0_2$ can also be described by limit operators. Define the limit operators $\lim_{\mathbf{\Sigma}^0_1}$, $\lim_{\mathbf{\Pi}^0_2}$, and $\lim_{\mathbf{\Sigma}^0_2}$ according to:

$$\begin{aligned}
\alpha \in \lim_{\mathbf{\Sigma}^0_1} f &\quad \text{iff} \quad \exists u \propto \alpha : f(u) = 1 \\
\alpha \in \lim_{\mathbf{\Pi}^0_2} f &\quad \text{iff} \quad \exists^\infty u \propto \alpha : f(u) = 1 \\
\alpha \in \lim_{\mathbf{\Sigma}^0_2} f &\quad \text{iff} \quad \forall^\infty u \propto \alpha : f(u) = 1
\end{aligned}$$

Here $\exists^\infty$ and $\forall^\infty$ mean "for infinitely many" and "for all but finitely many." It is not hard to prove that these limit operators exactly define the classes $\mathbf{\Sigma}^0_1$, $\mathbf{\Pi}^0_2$, and $\mathbf{\Sigma}^0_2$, respectively; see [11,30] for automata-theoretic proofs.

Using approximation functions of type $f : \omega^* \to \omega$, we introduce limit operators $\lim_{\mathbf{\Pi}^0_3}$ and $\lim_{\mathbf{\Sigma}^0_3}$:

$$\begin{aligned}
\alpha \in \lim_{\mathbf{\Pi}^0_3} f &\quad \text{iff} \quad \forall \ell : \forall^\infty u \propto \alpha : f(u) \neq \ell \quad \text{iff} \quad \liminf_{u \to \alpha} f(u) = \omega \\
\alpha \in \lim_{\mathbf{\Sigma}^0_3} f &\quad \text{iff} \quad \exists \ell : \exists^\infty u \propto \alpha : f(u) = \ell \quad \text{iff} \quad \liminf_{u \to \alpha} f(u) < \omega
\end{aligned}$$

where $u \to \alpha$ denotes that u takes the values $\langle\rangle, \langle\alpha_0\rangle, \langle\alpha_0, \alpha_1\rangle, \ldots$ for $\alpha = \alpha_0, \alpha_1, \ldots$

Proposition 7 Limit operators $\lim_{\mathbf{\Pi_3^0}}$ and $\lim_{\mathbf{\Sigma_3^0}}$ define the classes $\mathbf{\Pi_3^0}$ and $\mathbf{\Sigma_3^0}$; i.e. $S \in \mathbf{\Pi_3^0}$ if and only if there is an f such that $S = \lim_{\mathbf{\Pi_3^0}} f$, and $S \in \mathbf{\Sigma_3^0}$ if and only if there is an f such that $S = \lim_{\mathbf{\Sigma_3^0}} f$.

Proof We only give a proof for $\mathbf{\Sigma_3^0}$. The proof for $\mathbf{\Pi_3^0}$ is obtained by duality. A set $\lim_{\mathbf{\Sigma_3^0}} f$ is a $\mathbf{\Sigma_3^0}$ set, because it can be written as a countable union of $\mathbf{\Pi_2^0}$ sets:

$$\bigcup_i \lim_{\mathbf{\Pi_2^0}} f_i,$$

where

$$f_i(u) = \begin{cases} 1 & \text{if } f(u) = i \\ 0 & \text{if } f(u) \neq i \end{cases}$$

On the other hand, if S is a union $\bigcup_i \lim_{\mathbf{\Pi_2^0}} f_i$ of $\mathbf{\Pi_2^0}$ sets, then define

$$f(u) = \begin{cases} i & \text{if } i \text{ is minimal such that } f_i(u) = 1 \\ |u| & \text{if no such } i \text{ exists} \end{cases}$$

It follows that $\alpha \in \lim_{\mathbf{\Sigma_3^0}} f$ iff $\exists i : \exists^\infty u : f(u) = i$ iff $\exists i : \exists^\infty u : f_i(u) = 1$ iff $\exists i : \alpha \in \lim_{\mathbf{\Pi_2^0}} f_i$. □

7.3.2 Liminf Convergence Measure

Deterministic infinite-state Rabin automata exactly define the properties that are finite Boolean combinations of $\mathbf{\Pi_2^0}$ or $\mathbf{\Sigma_2^0}$ sets. In fact, it is not hard to show that the set accepted by a deterministic Rabin automaton A is the union of all sets of the form $\lim_{\mathbf{\Sigma_2^0}} f_R \cap \lim_{\mathbf{\Pi_2^0}} f_I$, where (R, I) is a Rabin pair of A, and $f_R(u)$ and $f_I(u)$ are expressed as in Section 7.2. Conversely, it can be shown that any Boolean combination of sets of the form $\lim_{\mathbf{\Sigma_2^0}} f$ can be encoded by a deterministic Rabin automaton A.

Lemma 10 shows that deterministic Rabin automata with infinite Rabin conditions define $\mathbf{\Sigma_3^0}$ sets; also, it is not hard to see that any $\mathbf{\Sigma_3^0}$ set can be defined by such an automaton. Thus by finding verification conditions for infinite Rabin conditions in Chapter 6, we extended the class of specifications that allow progress measures. But understanding these sets in terms of Rabin automata is less elegant than we would like.

Instead, we study the verification problem for programs defined by $\lim_{\mathbf{\Pi^0_3}}$ operators and specifications defined by $\lim_{\mathbf{\Sigma^0_3}}$ operators. This will also serve as a prelude to Chapter 8, where we extend progress measures to all analytic programs and coanalytic specifications. Consider the verification problem of determining whether $P \subseteq S$, where $P = \lim_{\mathbf{\Pi^0_3}} f_P$ and $S = \lim_{\mathbf{\Sigma^0_3}} f_S$, in terms of finite computations. We need a variation on Liminf measures that takes both f_P and f_S into account:

Definition 18 A *Liminf convergence measure* (μ, T) for (f_P, f_S) is a Liminf measure for $(G, u \mapsto \min\{f_P(u), f_S(u)\})$, where $G = (\omega^*, \{(u, u \cdot \langle n \rangle) \mid u \in \omega^*, n \in \omega\})$ is the infinite tree of finite computations.

A convergence measure guarantees that whenever a finite computation is lengthened, it converges towards being in S if it converges towards being in P. Formally we have the following characterization of the $P \subseteq S$ question:

Corollary 5 (of Theorem 4) For any $\mathbf{\Pi^0_3}$ set $P = \lim_{\mathbf{\Pi^0_3}} f_P$ and any $\mathbf{\Sigma^0_3}$ set $S = \lim_{\mathbf{\Sigma^0_3}} f_S$,

$$P \subseteq S \text{ iff } (f_P, f_S) \text{ has a Liminf convergence measure.}$$

Proof "$\Leftarrow$"Assume that (f_P, f_S) has measure (T, μ). Let $h(u) = \min\{f_P(u), f_S(u)\}$. Let $\alpha = \alpha_0, \alpha_1, \ldots \in P$. Then $\liminf_{u \to \alpha} f_P(u) = \omega$. Since $|\mu(v)| = h(v)$ and $\langle\rangle \to \langle\alpha_0\rangle \to \langle\alpha_0, \alpha_1\rangle \to \cdots$, we get a sequence $\mu(\langle\rangle) \trianglerighteq_{\mathrm{Li}} \mu(\langle\alpha_0\rangle) \trianglerighteq_{\mathrm{Li}} \mu(\langle\alpha_0, \alpha_1\rangle) \trianglerighteq_{\mathrm{Li}} \cdots$, so by Lemma 8, $\liminf_{u \to \alpha} \min\{f_P(u), f_S(u)\} < \omega$. We conclude $\liminf_{u \to \alpha} f_S(u) < \omega$, i.e. $\alpha \in S$.

"$\Rightarrow$" If $P \subseteq S$, then for all $\alpha \in \omega^\omega$, $\liminf_{u \to \alpha} \min\{f_P(u), f_S(u)\} < \omega$. Use Theorem 4 to obtain T and μ. □

7.3.3 Recursion-Theoretic Discussion

Assume that $(\phi_i)_i$ is an enumeration of the partial recursive functions from natural numbers to natural numbers. That ϕ_i converges on x is denoted $\phi_i(x)\downarrow$.

The computable closed sets can be enumerated as $\lim_{\mathbf{\Pi^0_1}} \phi_i$ based on the partial recursive functions $(\phi_i)_i$ and a fixed recursive encoding $\sigma : \omega^* \to \omega$ of finite sequences:

$$\alpha \in \lim_{\mathbf{\Pi^0_1}} \phi_i \text{ iff } \forall u \propto \alpha : (\phi_i(\sigma(u))\downarrow \Rightarrow \phi_i(\sigma(u)) = 1),$$

i.e. α is in the set defined by index i iff for all prefixes of α, if ϕ_i converges then its value is 1. The computable open sets can be represented:

$$\alpha \in \lim_{\mathbf{\Sigma^0_1}} \phi_i \quad \text{iff} \quad \exists u \propto \alpha : (\phi_i(\sigma(u))\downarrow \ \wedge \ \phi_i(\sigma(u)) = 1).$$

The other classes from Section 7.3.2 can be similarly represented. For these representations, total approximation functions are as expressive as partial functions:

Proposition 8 For each of the limit operators, total recursive approximation functions express the same class of sets as partial recursive functions.

Proof By dovetailing arguments [43], it can be seen that there is a uniform way of going from a partial recursive approximation function ϕ_i to a total recursive function $\phi_{r(i)}$ such that $\lim \phi_{r(i)} = \lim \phi_i$. □

Proposition 9 Checking whether an effectively represented graph G and a recursive height function h satisfy the Liminf condition is Π^1_1-complete.

Proof The problem is in Π^1_1 because it can be written as the formula:

$$\forall \text{ sequences } v_0, v_1, \ldots : (v_0, v_1, \ldots \text{ is path in } G \Rightarrow \exists \ell : \exists^\infty i : h(v_i) = \ell)$$

which consists of a second-order universal quantification "$\forall$ sequences $v_0, v_1, \ldots$" and a first-order formula "$v_0, v_1, \ldots$ is path in $G \Rightarrow \exists \ell : \exists^\infty i : h(v_i) = \ell$."

To show Π^1_1-hardness, we use a reduction from the classical Π^1_1-complete problem of determining that a recursive tree T only has finite paths [43]: for each node t in T, define $h(t) = |t|$; then the Liminf condition is satisfied iff T has no infinite paths. □

A more interesting issue is: given effective descriptions of G and h, is there an effective description of μ and T? Can it be calculated? The answers are yes, when G and h are presented in a sufficiently computable way, as the following discussion shows.

Definition 19 A graph $G = (V, E)$ is $*$-recursive if V, E, and $\rightarrow^*_W$, for all recursive W, is recursive.

The effective version of Lemma 9 is:

Lemma 9′ There is a recursive function that given an index of a nonempty countable $*$-recursive graph $G = (V, E)$ calculates the index of a (possibly infinite) recursive partition $\mathcal{S} = \{W_0, W_1, \ldots\}$ of V and the index of a recursive mapping $\iota : \omega \to \omega$ that to each number n gives an index of a node $\iota(n)$ in W_n (if W_n is defined) such that

- $\iota(W) \to_W^* W$; and
- for all $(v, v') \in E$, if $v \in W_k$ and $v' \in W_{k'}$, then $k \geq k'$.

Proof This follows from analysis of the proof of Lemma 9. □

Definition 20 We say that $\mu : V \to T$ is a *Liminf semi-measure* for (G, h) if $|\mu(v)| = h(v)$ and $(v, v') \in E$ implies $\mu(v) \trianglerighteq_{\mathrm{Li}} \mu(v')$.

In other words, a semi-measure is a measure, except that there is no requirement that T be finite-path.

Definition 21 We say that a Liminf measure for (G, h) can be obtained *uniformly* from indices of G and h if there is a recursive function m such that if i_G is an index for a graph G and i_h is an index for a height assignment h, then $m(i_G, i_h)$ is the index of recursive semi-measure μ; moreover μ is a measure for (G, h) (i.e. $T = \mathrm{rng}\mu$ is finite-path) iff $G \models \liminf h$.

The effective versions of Theorem 4 and Corollary 5 can now be formulated:

Theorem 4′ A Liminf measure for (G, h), where G is $*$-recursive and h is recursive, can be obtained uniformly from indices of G and h.

Proof This follows from analysis of the proof of Theorem 4. □

Corollary 5′ A Liminf progress measure for (P, S) can be obtained uniformly from indices of recursive f_P and f_S, where $P = \lim_{\mathbf{\Pi^0_3}} f_P$ and $S = \lim_{\mathbf{\Sigma^0_3}} f_S$.

Proof This follows from analysis of the proof of Theorem 5 and by the fact that the Baire space, viewed as a tree, is $*$-recursive. □

Corollary 5 implies that for an effectively presented program P and specification S, there is a finite argument, namely a recursive measure, for showing $P \subseteq S$. The

$P \subseteq Q$ question is Π_1^1-complete, because the even simpler problem of testing whether a given closed P is a subset of a given open S is Π_1^1-complete. To state this precisely, define $\mathrm{VER}(\Pi_1^0, \Sigma_1^0)$ to be the set of pairs (p, s) of indices such that p codes a closed set P, s codes an open set S, and $P \subseteq S$; define classes $\mathrm{VER}(\Sigma_1^0, \Pi_1^0)$, $\mathrm{VER}(\Pi_1^0, \Pi_1^0)$, $\mathrm{VER}(\Sigma_1^0, \Sigma_1^0)$ in the same way.

Proposition 10
(a) $\mathrm{VER}(\Sigma_1^0, \Pi_1^0)$ is Π_1^0-complete.
(b) $\mathrm{VER}(\Pi_1^0, \Sigma_1^0)$ is Π_1^1-complete.
(c) $\mathrm{VER}(\Pi_1^0, \Pi_1^0)$ is Π_1^1-complete.
(d) $\mathrm{VER}(\Sigma_1^0, \Sigma_1^0)$ is Π_1^1-complete.

Proof (a) To determine whether (p, s) is in $\mathrm{VER}(\Sigma_1^0, \Pi_1^0)$ amounts to checking that for any $\alpha \in \omega^\omega$, either $\alpha \notin \lim_{\mathbf{\Sigma_1^0}} \phi_p$ or $\alpha \in \lim_{\mathbf{\Pi_1^0}} \phi_s$. By Proposition 8 it can be assumed that ϕ_p and ϕ_s are recursive. But determining

$$\forall \alpha \in \omega^\omega : (\forall u \propto \alpha : \phi_p(u) \neq 1 \ \vee \ \forall u \propto \alpha : \phi_s(u) = 1)$$

is equivalent to determining

$$\forall u \in \omega^* : ((\forall u' \propto u : \phi_p(u') \neq 1) \ \vee \ (\forall u' \propto u : \phi_s(u') = 1))$$

which is in Π_1^0. The set $\mathrm{VER}(\Sigma_1^0, \Pi_1^0)$ is Π_1^0-hard; in fact, it is not difficult to see that the complement of the halting problem can be reduced to the question $\omega^\omega \subseteq \lim_{\Pi_1^0} \phi_p$.

The problems in (b), (c), and (d) are all in Π_1^1 because they can be written using one universal second-order quantifier. Moreover, the Π_1^1-complete problem of determining whether an effectively represented tree has the finite-path property can be reduced to the problem $\lim_{\Pi_1^0} \phi_p \subseteq \emptyset$. Thus (b), (c), and (d) are Π_1^1-hard. □

7.4 Automata-Theoretic Applications

Using the notion of Liminf condition we define a class of deterministic automata more expressive than Rabin automata. A deterministic *Liminf automaton* $A = (\Sigma, V, \rightarrow, \{v^0\}, h)$ is a deterministic automaton $A = (\Sigma, V, \rightarrow, \{v^0\})$ with a height assignment h. A run $v_0 \stackrel{e_0}{\rightarrow} v_1 \stackrel{e_1}{\rightarrow} \cdots$ of A over a behavior $e_0, e_1, \ldots$ is *accepting*

if $v_0, v_1, \ldots \models \liminf h$. The language $L(A)$ *accepted by* A is the set of behaviors whose run is accepting.

It is not hard to see using Proposition 7 that a set is $\mathbf{\Sigma_3^0}$ iff it is accepted by a deterministic Liminf automaton.

We can now formulate verification conditions for showing that $L(A_P) \subseteq L(A_S)$ where the program automaton $A_P = (\Sigma, V_P, \rightarrow_P, V_P^0)$ is nondeterministic and the specification automaton $A_S = (\Sigma, V_S, \rightarrow_S, \{s^0\}, h)$ is a deterministic Liminf automaton. Program and specification automata are related by means of a variant $\mathcal{I}$, which is a relation on $V_P \times V_S \times T$, where T is a finite-path pointer tree. The verification conditions for $\mathcal{I}$ are:

$(\mathrm{V1_{Li}})$ $\quad p \in Q_{\mathrm{II}}^0 \;\Rightarrow\; \exists t : \mathcal{I}(p, s^0, t)$

$(\mathrm{V2_{Li}})$ $\quad p \xrightarrow{e}_P p' \wedge \mathcal{I}(p,s,t) \;\Rightarrow\; \exists s', t' : s \xrightarrow{e}_S s' \wedge t \trianglerighteq_{\mathrm{Li}} t' \wedge \mathcal{I}(p', s', t')$

Corollary 6 (of Theorem 4) Let A_P be a nondeterministic automaton with no dead-end states and let A_S be a deterministic Liminf automaton. Then $L(A_P) \subseteq L(A_S)$ if and only if there is a variant $\mathcal{I}$ satisfying $(V1_{\mathrm{Li}})$ and $(V2_{\mathrm{Li}})$.

Proof The proof is nearly identical to the proof of Corollary 3 of Theorem 2. □

7.5 ¬Liminf Condition and Measure

We show that there are progress measures for the negated Liminf condition, ¬Liminf. It is satisfied on a graph G with height function h, written $G \models \neg \liminf h$, if $\liminf_{i \to \omega} h(v_i) = \omega$ for all infinite paths $v_0, v_1, \ldots$ in G. Note that G is necessarily acyclic if $G \models \neg \liminf h$. The Kleene-Brouwer ordering, in trees that are not finite-path, measures progress towards this condition:

Lemma 11 Let T be a pointer tree. If $t_0 \succ t_1 \succ \cdots$, then $\liminf_{k \to \omega} |t_k| = \omega$.

Proof It can be seen that $t_1[1] \geq t_2[1] \geq \cdots$, whence $t_i[1]$ is eventually constant. By an inductive argument it follows that for all ℓ, the value of $t_\ell[i]$ is eventually constant. Thus $\lim_{i \to \omega} t_i$ is an infinite path in T and $\liminf_{i \to \omega} |t_k| = \omega$. □

Definition 22 A *¬Liminf measure* (μ, T) for (G, h) is a pointer tree T and a mapping $\mu : G \to T$ such that

- for all v, $|\mu(v)| = h(v)$; and
- h respects the edge relation; i.e. if $(v, v') \in E$, then $\mu(v) \succ \mu(v')$.

¬Liminf measures do not constitute a complete verification method if the pointers are natural numbers. Informally, if natural numbers were enough, then the existence of a measure would be a Σ^1_1 question, contradicting that the fact that the ¬Liminf condition is Π^1_1-complete (which is seen by an easy reduction from the finite-path property).

Theorem 5 $G \models \neg \liminf h$ if and only if (G, h) has a ¬Liminf measure.

Proof "⇐" This direction follows from Lemma 11.

"⇒" Assume $G \models \neg \liminf h$. For each level ℓ, define a color set assignment (see Section 5.4)

$$CS^{\ell}(v) = \begin{cases} \{1\} & \text{if } h(v) > \ell \\ \emptyset & \text{if } h(v) \leq \ell \end{cases}$$

The color set assignment CS^{ℓ} is eventually permissible because $G \models \neg \liminf h$. By Lemma 7, there is an eventually stable coloring $c^{\ell} : V \to \{\bot, 1\}$ and a partition W^{ℓ}_{θ} of V. If $\ell \leq \mu(v)$, define $\mu(v)[\ell] = \theta$, where θ is such that $V \in W^{\ell}_{\theta}$. In this way a pointer tree T and a mapping $\mu : V \to T$ such that $|\mu(v)| = h(v)$ are specified. Now assume that $(u, v) \in E$. To prove that $\mu(u) \succ \mu(v)$, we let ℓ be the level of the least common ancestor of $\mu(u)$ and $\mu(v)$.

If $\ell = |\mu(u)| < |\mu(v)|$, then $\mu(u)$ is an ancestor of $\mu(v)$, thus $\mu(u) \succ \mu(v)$.

If $|\mu(v)| = \ell$, then $CS^{\ell}(v) = \emptyset$; thus $c^{\ell}(v) = \bot$, and by Lemma 7, u and v are in different θ^{ℓ} classes. This contradicts that ℓ is the least common ancestor.

Finally if $\ell < |\mu(u)|$ and $\ell < |\mu(v)|$, then by the assumption that ℓ is the level of the least common ancestor of $\mu(u)$ and $\mu(v)$, it is the case that $\mu(u)[\ell + 1] \neq \mu(v)[\ell + 1]$. Then by Lemma 7, $\mu(u)[\ell + 1] > \mu(v)[\ell + 1]$, whence $\mu(u) \succ \mu(v)$. □

7.6 Historical Notes

Harel showed that by transformations on trees representing programs one can do program verification for all finite levels of the Borel hierarchy [21]. The approach

in [15] is an adaptation of these ideas, based on modifying the program with explicit schedulers, to the programming language of loops. Vardi's study of automata theoretic techniques [50] is based on combining automata, letting specification automata specify incorrectness. This technique is also adopted by Manna and Pnueli [37].

Chapter 8

Coanalytic Specifications

We address the question: for which kind of programs and specifications can verification of infinite computations take place using measures on finite computations? No "reasonable" method (to be defined in Section 8.2.3) can exist for all programs and specifications that are analytic sets, which they are if given by nondeterministic infinite-state automata. Yet we show that for analytic P and coanalytic S, the question $P \subseteq S$ can be solved in terms of finite computations. In particular this verification theorem holds for any programs and specifications that are Borel sets. These form an important class that arises from infinitary temporal languages, for example.

The effective version of this result, which we call the Finite Argument Theorem, shows that there is a uniform way of obtaining a progress measure, i.e. an argument about finite computations, from P and S. From this follows the Suslin-Kleene Theorem: every set which is both analytic and coanalytic is Borel, and a Borel index can be obtained uniformly from an analytic and a coanalytic index.

In Section 8.1 we provide limit operators that allow deterministic definitions of analytic and coanalytic sets. In Section 8.2 we show that there exist progress measures for coanalytic specifications and obtain Suslin's Theorem as a corollary; we also state the Finite Argument Theorem. Next we introduce Borel automata in Section 8.3 as a means of defining Borel sets, and show the Boundedness Theorem for $\mathbf{\Pi}^1_1$ sets; we also show that Borel sets are analytic and coanalytic.

8.1 WF Limit Operators

For a sequence $\tau_0, \tau_1, \ldots$ of pointer trees over ω, we define $\lim_i \tau_i$ as:

$$t \in \lim_i \tau_i \text{ iff } \forall^\infty i : t \in \tau_i.$$

It is not hard to see that $\lim_i \tau_i$ is a tree, which we call the *limit tree*. To define infinite computations we use approximation functions τ that assign a finite tree $\tau(u)$ to each finite sequence $u \in \omega^*$. Let the limit operators $\lim_{\neg \mathrm{WF}}$ and $\lim_{\mathrm{WF}}$ be defined by: for $\alpha \in \omega^\omega$,

$$\alpha \in \lim_{\neg \mathrm{WF}} \tau \text{ iff } \neg \mathrm{WF} \lim_{u \to \alpha} \tau(u)$$

$$\alpha \in \lim_{\mathrm{WF}} \tau \text{ iff } \mathrm{WF} \lim_{u \to \alpha} \tau(u)$$

where $u \to \alpha$ denotes that u takes the values $\langle\rangle, \langle\alpha_0\rangle, \langle\alpha_0, \alpha_1\rangle, \ldots$ and $\mathrm{WF}\, T$ denotes that the tree T is well-founded, i.e. that it has the finite-path property.

The class $\mathbf{\Sigma^1_1}$ of analytic sets and the class $\mathbf{\Pi^1_1}$ of coanalytic sets were introduced in Section 7.3. They can also be described by limit operators:

Theorem 6 (Representation Theorem) The limit operators $\lim_{\neg \mathrm{WF}}$ and $\lim_{\mathrm{WF}}$ define the classes $\mathbf{\Sigma^1_1}$ and $\mathbf{\Pi^1_1}$, i.e. $S \in \mathbf{\Sigma^1_1} \Leftrightarrow \exists \tau : S = \lim_{\neg \mathrm{WF}} \tau$ and $S \in \mathbf{\Pi^1_1} \Leftrightarrow \exists \tau : S = \lim_{\mathrm{WF}} \tau$.

Proof We give the proof for $\lim_{\neg \mathrm{WF}}$ (the proof for $\lim_{\mathrm{WF}}$ is obtained by duality) and use the fact that a set S is analytic iff it is representable as a projection of a tree T whose pointers are in $\omega \times \omega$, i.e.

$$S = \{\alpha \in \omega^\omega \mid \exists \beta \in \omega^\omega : \forall n : (\alpha, \beta) \downarrow n \in T\}$$

(see e.g. [39, p.77]). Here $\downarrow n$ denotes the prefix up to index n, i.e. $(\alpha, \beta) \downarrow n = \langle (\alpha^0, \beta^0), \ldots, (\alpha^n, \beta^n) \rangle$.

Now for such a representation T of an analytic set S, define for all $u = \langle u^0, \ldots, u^n \rangle \in \omega^*$:

$$\tau(u) = \{\langle \beta^0, \ldots, \beta^n \rangle \mid \langle (u^0, \beta^0), \ldots, (u^n, \beta^n) \rangle \in T\}.$$

It follows that $\neg \mathrm{WF} \lim_{u \to \alpha} \tau(u)$ iff $\alpha \in S$.

On the other hand, if a set is given as $\lim_{\neg \mathrm{WF}} \tau$, then let T consist of all infinite paths $\langle (u^0, t^0), (u^1, t^1), \ldots \rangle$ such that $\langle t^0, t^1, \ldots \rangle \in \lim_{i \to \omega} \tau(\langle u^0, \ldots, u^i \rangle)$. Let S be the analytic set defined by T. Then we obtain that $\alpha \in \lim_{\neg \mathrm{WF}} \tau$ iff

$\neg \mathrm{WF}\lim_{u \to \alpha} \tau(u)$ iff $\alpha \in S$, i.e. $\lim_{\neg \mathrm{WF}} \tau$ is analytic. □

Note that this characterization of $\mathbf{\Sigma}^1_1$ and $\mathbf{\Pi}^1_1$ does not involve projection as did the representation used in the proof. As with the usual representations (see [39]), we have:

Theorem 7 (Boundedness Theorem for $\mathbf{\Pi}^1_1$ sets) Let $C = \lim_{\mathrm{WF}} \tau$ be a coanalytic set. If there is a countable ordinal ν such that for all $\alpha \in C$, $\lim_{u \to \alpha} \tau(u)$ is ν-rankable,[1] then C is Borel.

We postpone the proof of Theorem 7 to Section 8.3.

8.2 WF Condition, Relation, and Measure

We use tree embeddings to measure progress of sequences towards defining a finite-path tree in the limit. Let T be a fixed tree. An *embedding* of a tree τ in T is an injective mapping $\rho : \tau \to T$ such that $\rho(\langle\rangle) = \langle\rangle$ and for all $t \cdot \langle d \rangle \in \tau$ there is d' with $\rho(t \cdot \langle d \rangle) = \rho(t) \cdot \langle d' \rangle$. Note that $|\rho(t)| = |t|$; in fact, ρ is just a structure-preserving relabeling of τ. Also note that $\mathrm{dom}\rho$ is the tree τ and that $\mathrm{rng}\rho$ is the image in T of τ.

We can now define the *WF relation*, which we denote by $\trianglerighteq_{\mathrm{WF}}$:

Definition 23 (WF Relation) $\rho \trianglerighteq_{\mathrm{WF}} \rho'$ if for all $s \in \mathrm{dom}\rho \cap \mathrm{dom}\rho'$, $\rho(s) \succeq \rho'(s)$, where $\succeq$ is defined in Section 5.2.

Intuitively, $\rho \trianglerighteq_{\mathrm{WF}} \rho'$ holds if for any node s in both $\mathrm{dom}\rho$ and $\mathrm{dom}\rho'$, the image in T of s under ρ' is the same as or to the right of the image under ρ (assuming that pointer trees are depicted as in Figure 5.1). Although $\trianglerighteq_{\mathrm{WF}}$ is not a well-founded relation, it ensures well-foundedness in the limit provided T is well-founded.

Lemma 12 (WF Relation Lemma) If $\mathrm{WF}\,T$ and $\rho_0 \trianglerighteq_{\mathrm{WF}} \rho_1 \trianglerighteq_{\mathrm{WF}} \cdots$, then $\mathrm{WF}\lim_i \mathrm{dom}\rho_i$.

This lemma is an immediate consequence of:

[1] Here, a well-founded tree T is *ν-rankable* if there is an assignment of ordinals to the nodes of T such that the root has rank ν and if a node t has rank γ then every child of t has rank less than γ.

Proposition 11 Let T be a fixed tree and let $\rho_0 \trianglerighteq_{\mathrm{WF}} \rho_1 \trianglerighteq_{\mathrm{WF}} \cdots$ be an infinite $\trianglerighteq_{\mathrm{WF}}$-related sequence of embeddings in T. Then there is an embedding ρ of $\lim_i \mathrm{dom}\rho_i$ in $\lim_i \mathrm{rng}\rho_i$.

Proof Let $s = \langle s^0, \ldots, s^\ell \rangle \in \lim_i \mathrm{dom}\rho_i$. Then there is K such that for all $k \geq K$, $s \in \mathrm{dom}\rho_k$. Hence by definition of $\trianglerighteq_{\mathrm{WF}}$, we have $\rho_K(\langle s^0, \ldots, s^\ell \rangle) \succeq \rho_{K+1}(\langle s^0, \ldots, s^\ell \rangle) \succeq \cdots$. Thus by the Kleene-Brouwer Ordering Lemma (Section 5.2), there is a $H \geq K$ and directions $t^0, \ldots, t^\ell$ such that for all $k \geq H$, $\rho_k(\langle s^0, \ldots, s^\ell \rangle) = \langle t^0, \ldots, t^\ell \rangle$. Therefore $\langle t^0, \ldots, t^\ell \rangle$ is in $\lim_i \mathrm{rng}\rho_i$. Now define $\rho(t) = \langle t^0, \ldots, t^\ell \rangle$.

It is not hard to verify that ρ so defined is an embedding of $\lim_i \mathrm{dom}\rho_i$ in $\lim_i \mathrm{rng}\rho_i$. □

Hence if T is well-founded, $\trianglerighteq_{\mathrm{WF}}$ measures progress of trees towards defining a well-founded tree. To state this more forcefully, we need some definitions.

Definition 24 A *tree assignment* τ on a graph $G = (V, E)$ is a mapping τ that defines a finite tree $\tau(v)$ for each node in V. An infinite path $v_0, v_1, \ldots$ satisfies the *WF condition* with respect to τ, written $v_0, v_1, \ldots \models \mathrm{WF}\tau$, if $\mathrm{WF}\lim_i \tau(v_i)$. A graph G satisfies the *WF condition*, written $G \models \mathrm{WF}\tau$, if every infinite path in G satisfies the WF condition.

Definition 25 A *WF measure* (μ, T) for (G, τ) is a finite-path tree T and a mapping $\mu : v \in V \rightarrow (\tau(v) \rightarrow T)$ such that

- $\mu(v)$ is an embedding of $\tau(v)$ in T; and
- μ respects the edge relation of G, i.e. $(u, v) \in E$ implies $\mu(u) \trianglerighteq_{\mathrm{WF}} \mu(v)$.

A key result of this chapter is:

Theorem 8 $G \models \mathrm{WF}\tau$ iff (G, τ) has a WF measure.

Proof "⇐" Let (μ, T) be a WF measure for (G, τ) and let $v_0, v_1, \ldots$ be an infinite path in G. Then $\mu(v_0) \trianglerighteq_{\mathrm{WF}} \mu(v_1) \trianglerighteq_{\mathrm{WF}} \cdots$. By the WF Relation Lemma, $\mathrm{WF}\lim_i \mathrm{dom}\mu(v_i)$, and since $\mathrm{dom}\mu(v_i) = \tau(v_i)$, it follows that $v_0, v_1, \ldots \models \mathrm{WF}\tau$.

"⇒" The construction is similar to the construction in the proof of Theorem 4 in Chapter 7. Given that $G \models \mathrm{WF}\tau$, we show how to construct a finite-path tree T and a measure μ in T. We use the algorithm in Figure 8.1. For now, T is

AssignWF(t):

For each d:

(a) $\widehat{W} := \{v \in W(t) \mid (\wp_2 t) \cdot \langle d \rangle \in \tau(v)\}$.
If $\widehat{W} = \emptyset$, then skip (b) and (c).

(b) Use Lemma 9 to obtain a partition $\mathcal{S} = W_0, W_1, \ldots$ of $\widehat{W}$.

(c) For each class W_k of $\mathcal{S}$:

i. $T := T \cup \{t \cdot \langle (k,d) \rangle\}$.

ii. $W(t \cdot \langle (k,d) \rangle) := W_k$.

iii. **AssignWF**($t \cdot \langle (k,d) \rangle$).

Figure 8.1: Algorithm **AssignWF**.

defined as a tree with directions in $\omega \times \omega$, but later we view it as a tree on ω by using an appropriate pairing function. Each node $t = \langle (k^0, s^0), \ldots, (k^n, s^n) \rangle$ of T has a label $W(t)$, which is a subset of V constructed so that for all $v \in W(t)$, $\langle s^0, \ldots, s^n \rangle \in \tau(v)$. The algorithm **AssignWF** is initially applied with $W(\langle\rangle) = V$, $t = \langle\rangle$, and $T = \{\langle\rangle\}$.

The purpose of **AssignWF**(t) is to define the children of node t together with their labels. In Step (a) of Figure 8.1, nodes v in $W(t)$ whose tree $\tau(v)$ does not contain $\langle s^0, \ldots, s^n, d \rangle$ are filtered out; here $\wp_2(\langle (k^0, s^0), \ldots, (k^n, s^n) \rangle) = \langle s^0, \ldots, s^n \rangle$. In Step (b) the resulting set $\widehat{W}$ is partitioned using Lemma 9. In Step (c) a child in direction (k,d) with label W_k is created for each class W_k of the partition and the algorithm is reapplied on W_k and node $t \cdot \langle k, d \rangle$.

We take the tree T to be the value of variable T calculated by the transfinite applications of **AssignWF**(t) starting from the root.

Note that Lemma 9 associates to each label $W(t)$, $t \neq \langle\rangle$, a node $\iota(W(t))$ of G such that $\iota(W(t)) \rightarrow^*_{W(t)} W(t)$.

Claim 10

(a) For all $t \in T$, for all $v \in W(t)$, $\wp_2(t) \in \tau(v)$.

(b) For any $t \cdot \langle (k,d) \rangle \in T$, $W(t) \supseteq W(t \cdot \langle (k,d) \rangle)$.

(c) For each $v \in V$ and each $\langle s^0, \ldots, s^n \rangle \in \tau(v)$, there is a unique list $\langle k^0, \ldots, k^n \rangle$ of length n such that $v \in W(\langle (k^0, s^0), \ldots, (k^n, s^n) \rangle)$.

Proof The proof is by induction. □

Now using Claim 10(c), we define an embedding $\mu(v)$ of $\tau(v)$ in T by $\mu(v)(s) = \langle (k^0, s^0), \ldots, (k^n, s^n) \rangle$ for all $s = \langle s^0, \ldots, s^n \rangle \in \tau(v)$. To prove that μ preserves order, assume that $(u, v) \in E$ and $s = \langle s^0, \ldots, s^n \rangle \in \mathrm{dom}\tau(u) \cap \mathrm{dom}\tau(v)$. Let $k^0, \ldots, k^n, h^0, \ldots, h^n$ be defined by:

$$\begin{aligned} \langle (k^0, s^0), \ldots, (k^n, s^n) \rangle &= \mu(u)(\langle s^0, \ldots, s^n \rangle) \\ \langle (h^0, s^0), \ldots, (h^n, s^n) \rangle &= \mu(v)(\langle s^0, \ldots, s^n \rangle) \end{aligned}$$

Then either $\langle k^0, \ldots, k^n \rangle = \langle h^0, \ldots, h^n \rangle$ or there is a $\lambda \leq n$ such that $k^\lambda \neq h^\lambda$ and $k^\ell = h^\ell$ for all $\ell < \lambda$. Let

$$t = \langle (k^0, s^0), \ldots, (k^{\lambda-1}, s^{\lambda-1}) \rangle = \langle (h^0, s^0), \ldots, (h^{\lambda-1}, s^{\lambda-1}) \rangle.$$

Then $\texttt{AssignWF}(t)$ was applied and $u, v \in W(t)$. This application defined $W(t \cdot \langle (k^\lambda, s^\lambda) \rangle)$ and $W(t \cdot \langle (h^\lambda, s^\lambda) \rangle)$, which are disjoint subsets of $W(t)$ by the assumption that $k^\lambda \neq h^\lambda$. Hence $k^\lambda > h^\lambda$ by Lemma 9. Therefore, if we use a pairing function $\# : \omega \times \omega \to \omega$ that is monotonic in its first variable to code the directions of T, we obtain

$$\langle \#(k^0, s^0), \ldots, \#(k^n, s^n) \rangle \geq \langle \#(h^0, s^0), \ldots, \#(h^n, s^n) \rangle.$$

Since this holds for all $s \in \mathrm{dom}\tau(u) \cap \mathrm{dom}\tau(v)$, it follows that $\mu(u) \trianglerighteq_{\mathrm{WF}} \mu(v)$.

Claim 11 There is no infinite path in T.

Proof Suppose for a contradiction that $\langle (k^0, s^0), (k^1, s^1), \ldots \rangle$ is an infinite path. Then by Claim 10(b), $W() \supseteq W(\langle (k^0, s^0), (k^1, s^1) \rangle \supseteq \cdots$ and therefore for any $n > 0$, if $t = \langle (k^0, s^0), \ldots, (k^n, s^n) \rangle$ then $\iota(W(t)) \to^*_{W(t)} W(t \cdot \langle (k^{n+1}, s^{n+1} \rangle)$, because $\iota(W(t)) \to^*_{W(t)} W(t)$ by Lemma 9. In particular, $\iota(W(t)) \to^*_{W(t)} \iota(W(t \cdot \langle (k^{n+1}, s^{n+1}) \rangle))$. We conclude that there exists an infinite path $v_0, v_1, \cdots$ in G and a monotonically increasing function $i(j)$ such that

$$v_{i(j)} = \iota(W(\langle (k^0, s^0), \ldots, (k^j, s^j) \rangle))$$

for all j, and for all $n \geq i(j)$, Claim 10(a) implies that $\langle s^0, \ldots, s^j \rangle \in \tau(v_n)$. But then $v_0, v_1, \cdots$ does not satisfy the WF condition, because $\langle s^0, s^1, \ldots \rangle$ is an infinite path in $\lim_i \tau(v_i)$. This is a contradiction. □

8.2.1 WF Convergence Measure

Let $P = \lim_{\neg \mathrm{WF}} f_P$ be an analytic program and $S = \lim_{\mathrm{WF}} f_S$ a coanalytic specification. To prove that $P \subseteq S$ in terms of finite computations, we need a way of braiding trees of approximation functions.

Definition 26 Given finite trees τ and τ', the set $M^\vee(\tau, \tau')$ consisting of nodes $\langle d^0, e^0, \ldots, d^n, e^n \rangle$ and $\langle d^0, e^0, \ldots, e^{n-1}, d^n \rangle$, where $n \geq -1$, $\langle d^0, \ldots, d^n \rangle \in \tau$, and $\langle e^0, \ldots, e^n \rangle \in \tau'$, is called the *or-merge* of τ and τ'.

It is not hard to see that $M^\vee(\tau, \tau')$ is a tree. The or-merge has the following properties:

Proposition 12 Let τ_i and τ'_i be infinite sequences of trees.

(a) $\mathrm{WF}\lim_i M^\vee(\tau_i, \tau'_i)$ iff $\mathrm{WF}\lim_i \tau_i$ or $\mathrm{WF}\lim_i \tau'_i$.

(b) If $\lim M^\vee(\tau_i, \tau'_i)$ is ν-rankable and $\neg\mathrm{WF}\lim_i \tau_i$, then $\lim_i \tau'_i$ is ν-rankable.

Proof (a)"$\Rightarrow$" Assume that $\neg\mathrm{WF}\lim_i \tau_i$ and $\neg\mathrm{WF}\lim_i \tau'_i$. Then there are infinite paths $\langle d^0, d^1, \ldots \rangle$ and $\langle e^0, e^1, \ldots \rangle$ in $\lim_i \tau_i$ and $\lim_i \tau'_i$, and $\langle d^0, e^0, d^1, e^1, \ldots \rangle$ is an infinite path in $\lim_i M^\vee(\tau_i, \tau'_i)$.

(a)"$\Leftarrow$" Assume that $\langle d^0, e^0, d^1, e^1, \ldots \rangle$ is an infinite path in $\lim_i M^\vee(\tau_i, \tau'_i)$. Then $\langle d^0, d^1, \ldots \rangle$ and $\langle e^0, e^1, \ldots \rangle$ are infinite paths in $\lim_i \tau_i$ and $\lim_i \tau'_i$.

(b) Assume $\lim M^\vee(\tau_i, \tau'_i)$ is ν-rankable and $\neg\mathrm{WF}\lim_i \tau_i$. Then there is an infinite path $\langle d^0, d^1, \ldots \rangle$ in $\lim_i \tau_i$. For every $\langle e^0, \ldots, e^n \rangle$ in $\lim_i \tau'_i$, $\langle d^0, e^0, \ldots, d^n, e^n \rangle$ is in $\lim_i M^\vee(\tau_i, \tau'_i)$. Now let $\gamma(t)$ be the rank of $\lim_i M^\vee(\tau_i, \tau'_i)$ in a ν-ranking of $\lim M^\vee(\tau_i, \tau'_i)$, and define a ranking γ' of $\lim_i \tau'_i$ by

$$\gamma'(\langle e^0, \ldots, e^n \rangle) = \gamma(\langle d^0, e^0, \ldots, d^n, e^n \rangle).$$

Then γ' is a ν-ranking of $\lim_i \tau'_i$. □

To measure finite computations of $P = \lim_{\neg \mathrm{WF}} \tau_P$ and $S = \lim_{\mathrm{WF}} \tau_S$, we need:

Definition 27 A *WF convergence measure* (T, μ) for (τ_P, τ_S) is a WF measure for $(G, u \mapsto M^\vee(\tau_P(u), \tau_S(u))$, where $G = (\omega^*, \{(u, u \cdot \langle n \rangle) \mid u \in \omega^*, n \in \omega\})$ is the infinite tree of finite computations.

The main result of this chapter is:

Corollary 7 (of Theorem 8) For analytic $P = \lim_{\neg \mathrm{WF}} \tau_P$ and coanalytic $S = \lim_{\mathrm{WF}} \tau_S$,

$$P \subseteq S \text{ iff } (\tau_P, \tau_S) \text{ has a WF convergence measure.}$$

Proof "$\Leftarrow$" Assume that (τ_P, τ_S) has measure (T, μ). Let $\alpha \in P$, i.e. $\neg \mathrm{WF} \lim_{u \to \alpha} \tau_P(u)$. Because μ is a WF measure and $\langle\rangle, \langle\alpha_0\rangle, \langle\alpha_1\rangle, \langle\alpha_1, \alpha_2\rangle \cdots$ is an infinite path in G, we get an infinite $\trianglerighteq_{\mathrm{WF}}$-related sequence

$$\mu(\langle\rangle) \trianglerighteq_{\mathrm{WF}} \mu(\langle\alpha_0\rangle) \trianglerighteq_{\mathrm{WF}} \mu(\langle\alpha_0, \alpha_1\rangle) \trianglerighteq_{\mathrm{WF}} \cdots$$

Thus by Definition 27 and the WF Relation Lemma, $\mathrm{WF} \lim_{u \to \alpha} M^{\vee}(\tau_P(u), \tau_S(u))$. Using Proposition 12(a), we deduce that $\mathrm{WF} \lim_{u \to \alpha} \tau_S(u)$, whence $\alpha \in S$.

"$\Rightarrow$" If $P \subseteq S$, then by Proposition 12(a) it holds for all $\alpha \in \omega^\omega$ that

$$\mathrm{WF} \lim_{u \to \alpha} M^{\vee}(\tau_P(u), \tau_S(u)).$$

Then T and μ can be obtained from Theorem 8. □

8.2.2 Suslin's Theorem

Corollary 8 (Suslin's Theorem) If P is analytic and coanalytic, then P is Borel.

Proof If P is analytic, i.e. $P = \lim_{\neg \mathrm{WF}} \tau_P$ for some τ_P, and coanalytic, i.e. $P = \lim_{\mathrm{WF}} \tau_S$ for some τ_S, then by Corollary 7 (τ_P, τ_S) has a convergence measure (μ, T), because $P \subseteq P$. Let ν be a countable ordinal such that T is ν-rankable. Then by Proposition 12(b), for any $\alpha \in P$, $\lim_{u \to \alpha} \tau_S(u)$ is ν-rankable, because $\neg \mathrm{WF} \lim_{u \to \alpha} \tau_P(u)$. Then P is Borel by Theorem 7. □

8.2.3 Finite Argument Theorem

A measure can be viewed as an argument for why a program satisfies a specification. We show that for effective descriptions of the program and the specification there is an effective description of the measure. More precisely, there is a total recursive function calculating an index for a semi-measure μ given indices for the program P and the specification S; moreover, μ is a measure iff P satisfies S. Here a semi-measure (μ, T) is as defined in Chapter 7, i.e. it is a measure except that there

is no requirement that T be finite-path. We state the following results without proofs, which rely on the same techniques as those of Chapter 7.

Theorem 8′ A WF measure for (G, τ), where G is $*$-recursive and τ is recursive, can be obtained uniformly from indices of G and τ.

As with the representations in Chapter 7, effectively computable analytic sets can be represented by some suitable enumeration τ_i of total recursive functions, where the value of $\tau_i(u)$ for $u \in \omega^*$ is a finite tree.

Corollary 7′ (Finite Argument Theorem) A WF convergence measure for (P, S) can be obtained uniformly from indices of τ_p and τ_s, where $P = \lim_{\neg \mathrm{WF}} \tau_p$ and $S = \lim_{\mathrm{WF}} \tau_s$.

Intuitively, the Finite Argument Theorem shows that there is a systematic (in fact computable) way of getting a finite argument of correctness about finite computations from the program and the specification. It can be formulated as a verificationist slogan: assertional methods are always applicable! The Finite Argument Theorem shows that the set $\mathrm{VER}(\Sigma_1^1, \Pi_1^1)$ (i.e. the set of pairs (p, s) coding recursive approximation functions τ_p and τ_s such that $\lim_{\neg \mathrm{WF}} \tau_p \subseteq \lim_{\mathrm{WF}} \tau_s$) is in Π_1^1; since $\mathrm{VER}(\Pi_1^0, \Sigma_1^0)$ is Π_1^1-hard (Proposition 10), it follows that $\mathrm{VER}(\Sigma_1^1, \Pi_1^1)$ is Π_1^1-complete.

The verification method based on WF convergence measures is optimal in the following sense. For the other natural extension of Borel sets, namely Σ_1^1, the verification problem $\mathrm{VER}(\Sigma_1^1, \Sigma_1^1)$ is Π_2^1-complete. For example, determining whether $L(A_P) \subseteq L(A_S)$, where A_P and A_S are nondeterministic automata, is Π_2^1-complete [47] (nondeterministic automata define analytic sets; this can be argued using the projection representation of Theorem 6; see also [11]). It is hardly imaginable that a reasonable verification method would not be in Σ_2^1, which allows one to guess relations and verify that they are well-founded. But even a Σ_2^1 method cannot possible solve the Π_2^1-complete verification problem for Σ_1^1 sets. In this sense the results of this chapter are optimal.

Finally we observe that, just as Suslin's Theorem is a consequence of Corollary 7, the Finite Argument Theorem implies Kleene's Theorem, which states that there is a uniform way of obtaining an index in the hyperarithmetical hierarchy of a set P from a Π_1^1 and a Σ_1^1 index of P.

8.3 Borel Automata

To describe Borel sets in terms that are useful for verification of programs, we introduce a class of automata whose acceptance conditions are infinitary temporal logic formulas. This will also allow us to proof the Boundedness Theorem for $\mathbf{\Pi^1_1}$ sets.

Definition 28 By transfinite induction, we define a *ranked formula* ϕ_γ, where γ is a countable ordinal, to be either a *temporal predicate* $\Box\Diamond\Phi$ ("infinitely often Φ") or $\Diamond\Box\Phi$ ("almost always Φ"), where Φ is a predicate on V, or a disjunction $\bigvee_{\gamma'<\gamma}\phi_{\gamma'}$ or a conjunction $\bigwedge_{\gamma'<\gamma}\phi_{\gamma'}$, where $\phi_{\gamma'}$ are ranked formulas.

A sequence $v_0, v_1, \ldots$ *satisfies* ϕ_γ, written $v_0, v_1, \ldots \models \phi_\gamma$, according to:

$$v_0, v_1, \ldots \models \Diamond\Box\Phi \text{ iff } \exists H : \forall h > H : v_h \models \Phi$$

$$v_0, v_1, \ldots \models \Box\Diamond\Phi \text{ iff } \forall H : \exists h > H : v_h \models \Phi$$

$$v_0, v_1, \ldots \models \bigwedge_{\gamma'<\gamma} \phi_{\gamma'} \text{ iff } \forall \gamma' < \gamma : v_0, v_1, \ldots \models \phi_{\gamma'}$$

$$v_0, v_1, \ldots \models \bigvee_{\gamma'<\gamma} \phi_{\gamma'} \text{ iff } \exists \gamma' < \gamma : v_0, v_1, \ldots \models \phi_{\gamma'}$$

Definition 29 A *Borel automaton* $A = (V, \rightarrow, \{v^0\}, \phi_\nu)$ on alphabet ω consists of a countable set of states V, a deterministic transition relation $\rightarrow \subseteq V \times \omega \times V$, an initial state v^0, and a ranked formula ϕ_ν.

We assume that A allows a run (which is unique because A is deterministic) over every $\alpha \in \omega^\omega$. A run $v_0 \stackrel{\alpha_0}{\rightarrow} v_1 \stackrel{\alpha_1}{\rightarrow} \cdots$ over $\alpha = \alpha_0, \alpha_1, \ldots$ is *accepting* if $v_0, v_1, \ldots \models \phi_\nu$. The language $L(A)$ *accepted by* A is the set of $\alpha \in \omega^\omega$ whose run is accepting.

Proposition 13 The class of languages defined by Borel automata is the class of Borel sets.

Proof Automata of the form $(V, \rightarrow, \{v^0\}, \Box\Diamond\Phi)$ are deterministic Büchi automata, which define the class $\mathbf{\Pi^0_2}$ [11,30]. The rest of the proof is by straightforward transfinite induction. □

8.3.1 Proof of the Boundedness Theorem of Section 8.1

We assume that $S = \lim_{\mathrm{WF}} \tau_S$ is a coanalytic set and ν is a countable ordinal such that for all $\alpha \in S$, $\lim_{u \to \alpha} \tau_S(u)$ is ν-rankable. We define a Borel automaton with $V = \{(u, \tau_S(u)) \mid u \in \omega^*\}$, $v^0 = (\langle\rangle, \tau_S(\langle\rangle))$, and transition relation $(u, \tau) \xrightarrow{d} (u', \tau')$ if $u' = u \cdot \langle d \rangle$. By transfinite induction, define ϕ^t_γ, where $t \in \omega^*$, as

$$\begin{aligned} \phi^t_0 &= \bigwedge_{d \in \omega} \Box\Diamond\neg\Phi^{t \cdot \langle d \rangle} \\ \phi^t_\gamma &= \bigwedge_{d \in \omega} (\Box\Diamond\neg\Phi^{t \cdot \langle d \rangle} \vee \bigvee_{\gamma' < \gamma} \phi^{t \cdot \langle d \rangle}_{\gamma'}), \quad \text{if } \gamma > 0 \end{aligned}$$

where Φ^t is a predicate on V defined by $(u, \tau) \models \Phi^t$ iff $t \in \tau$. It is not hard to see that $(u_0, \tau_0), (u_1, \tau_1), \ldots \models \phi^t_\gamma$ iff the limit of $\tau_0[t], \tau_1[t], \ldots$ is γ-rankable, where $\tau[t]$ denotes the empty tree if $t \notin \tau$ and otherwise denotes the subtree of τ rooted at t.

Hence, if we let $A = (V, \to, \{v^0\}, \phi^{\langle\rangle}_\nu)$, then $L(A) = S$, whence S is Borel by Proposition 13.

8.3.2 Borel Sets Are Analytic and Coanalytic

We show how to translate the temporal logic acceptance condition of a Borel automaton into an approximation function. This means that program verification with temporal logic can take place by measuring progress using Theorem 8. The translation also proves that all Borel sets are analytic and coanalytic.

Theorem 9 Let $A = (V, \to, \{v^0\}, \phi_\nu)$ be a Borel automaton. There exist approximation functions τ and τ' such that

$$\lim_{\mathrm{WF}} \tau = L(A)$$

$$\lim_{\neg\mathrm{WF}} \tau' = L(A)$$

In order to prove this theorem we need two lemmas. They show how to merge countably many sequences of finite trees into one such sequence that satisfies the WF condition iff all (respectively, one) of the original sequences satisfy the WF condition.

Lemma 13 There is an operation $m^\forall$ that merges any list of finite trees into a finite tree such that for any collection $(\tau_i^j)_i$, $j \in \omega$, of sequences of finite trees:

$$\begin{aligned} &\text{WF } \lim_{i\to\omega} m^\forall(\tau_0^i, \ldots, \tau_i^i) \\ &\text{iff} \\ &\forall j : \text{WF } \lim_{i\to\omega} \tau_i^j \end{aligned}$$

Proof Let $\langle i\rangle \cdot \tau$ denote the set $\{\langle i\rangle \cdot v \mid v \in \tau\}$, i.e. the tree τ grafted on a trunk $\langle i\rangle$. Let

$$m^\forall(\tau^0, \ldots, \tau^i) = \{\langle\rangle\} \cup \bigcup_{h\le i} \langle h\rangle \cdot \tau^h.$$

We observe that for all $i \ge j$, $\langle j\rangle \in m^\forall(\tau_i^0, \ldots, \tau_i^i)$ and the subtree rooted at $\langle j\rangle$ is τ_i^j. The lemma follows from this observation. □

Lemma 14 There is an operation $m^\exists$ that merges any list of finite trees into a finite tree such that for any collection $(\tau_i^j)_i$, $j \in \omega$, of sequences of finite trees:

$$\begin{aligned} &\text{WF } \lim_{i\to\omega} m^\exists(\tau_0^i, \ldots, \tau_i^i) \\ &\text{iff} \\ &\exists j : \text{WF } \lim_{i\to\omega} \tau_i^j \end{aligned}$$

Proof The merge tree $\tau = m^\exists(\tau^0, \ldots, \tau^i)$ is obtained by braiding the trees $\tau^0, \ldots, \tau^i$ so that each level ℓ of τ corresponds to nodes of the tree $\tau^{\pi(i)}$; here $\pi : \omega \to \omega$ is a function such that for all ℓ, $\pi(\ell) \le \ell$ and π takes on every value infinitely often. The pointers of τ are pairs (χ, ξ), where χ represents the tree τ^χ and ξ represents a pointer of the tree τ^χ. Thus nodes of τ are of the form $\langle(\chi_0, \xi_0), \ldots, (\chi_n, \xi_n)\rangle$. Although the pointers of τ are pairs, they can be converted to numbers by some encoding.

We define a function h^χ to retrieve nodes of color χ from τ and a function $\#^\chi$, which computes the level in τ^χ of the node retrieved. For any χ and any $t \in \tau$, let

$$h^\chi(t) = \langle \xi_{\ell(0)}, \ldots, \xi_{\ell(N)}\rangle$$

$$\#^\chi(t) = N + 1$$

where $t = \langle(\chi_0, \xi_0), \ldots, (\chi_n, \xi_n)\rangle$ and $N \ge -1$ and $\ell : [0..N] \to [0..n]$ are such that $\ell(0), \ldots, \ell(N)$ designate the indices, in increasing order, of all the pairs of color χ in t.

$\exists\texttt{Merge}(\tau^0, \ldots, \tau^i)$:

$\tau := \{\langle\rangle\}$
For $\ell = 1$ to i
$\chi = \pi(\ell)$
For all t of τ at level $\ell - 1$
For all ξ such that $h^\chi(t) \cdot \langle\xi\rangle \in \tau^\chi$:
$\tau := \tau \cup \{t \cdot \langle(\chi, \xi)\rangle\}$

Figure 8.2: Algorithm $\exists\texttt{Merge}$.

In Figure 8.2 it is shown how the merge tree τ is constructed. Note that when a node $t \cdot \langle(\chi, \xi)\rangle$ is added to τ, it is the case that $h^\chi(t) \cdot \langle\xi\rangle \in \tau^\chi$, therefore $h^\chi(t \cdot \langle(\chi, \xi)\rangle) = h^\chi(t) \cdot \langle\xi\rangle \in \tau^\chi$.

We prove the "$\Leftarrow$" direction of the lemma by contraposition. Assume that $P = \langle(\chi_0, \xi_0), (\chi_1, \xi_1), \ldots\rangle$ is an infinite path in $\lim_{i\to\omega} m^\exists(\tau_0^i, \ldots, \tau_i^i)$. For all χ, define $h^\chi(P) = \langle\xi^\chi_{\ell(0)}, \xi^\chi_{\ell(1)}, \ldots\rangle$, where $\ell(0), \ell(1), \ldots$ as before designate the indices of the pairs of color χ in P. Note that since $\pi(\ell)$ takes the value χ infinitely often, the function ℓ is indeed strictly increasing. Then for all χ, $\langle\xi^\chi_{t(0)}, \xi^\chi_{t(1)}, \ldots\rangle$ is an infinite path in $\lim_i(\tau^j)_i$. Thus, for all j, $\neg\text{WF } \lim_{i\to\omega} \tau_i^j$.

For the "$\Rightarrow$" direction, the proof is also by contraposition. Assume that for all j, $\neg\text{WF } \lim_{i\to\omega} \tau_i^j$. Then for each j, there is an infinite path $\langle\xi_0^j, \xi_1^j, \ldots\rangle$ in $\lim_i \tau_i^j$. Now, for $\ell = 1, 2, \ldots$, let $\chi_\ell = \pi(\ell)$ and let $\xi_\ell = \xi^h_{\chi_\ell}$, where $h = \#^{\chi_\ell}(\langle(\chi_0, \xi_0), \ldots, (\chi_{\ell-1}, \xi_{\ell-1})\rangle)$. It can be seen that $\langle(\chi_0, \xi_0), (\chi_1, \xi_1), \ldots\rangle$ is an infinite path in $\lim_{i\to\omega} m^\exists(\tau_0^i, \ldots, \tau_i^i)$. □

Proof (Theorem 9) Given $A^\phi = (V, \to \{v^0\}, \phi)$, we construct by structural induction on ϕ an approximation function τ such that $\lim_{\text{WF}} \tau = L(A^\phi)$.

For an automaton $A^{\Box\Diamond\Phi}$, define

$$\tau^{\Box\Diamond\Phi}(u) = \begin{cases} \langle\rangle & \text{if } v_{n+1} \models \Phi \\ \underbrace{\langle 0, \ldots, 0\rangle}_{n} & \text{if } v_{n+1} \not\models \Phi \end{cases}$$

where $v_0, \ldots, v_{n+1}$ are the states of the partial run $v_0 \xrightarrow{\alpha_0} \cdots \xrightarrow{\alpha_n} v_{n+1}$ over $u = \alpha_0, \ldots, \alpha_n$. Then

$$\lim_{u \to \alpha} \tau^{\Box\Diamond\Phi} = \begin{cases} \langle\rangle & \text{if } v_0, v_1, \ldots \models \Box\Diamond\Phi \\ \langle 0, 0, \ldots \rangle & \text{if } v_0, v_1, \ldots \not\models \Box\Diamond\Phi \end{cases}$$

i.e. $\alpha \in \lim_{\mathrm{WF}} \tau^{\Box\Diamond\Phi}$ iff $\alpha \in L(A^{\Box\Diamond\Phi})$.

For an automaton $A^{\Diamond\Box\Phi}$, define

$$\tau^{\Diamond\Box\Phi}(u) = \begin{cases} \langle\rangle & \text{if for all } m \leq n,\ v_m \models \Phi \\ \underbrace{\langle 0, \ldots, 0 \rangle}_{m} & \text{if } m \leq n \text{ is a maximal index such that } v_m \not\models \Phi \end{cases}$$

Then $\alpha \in \lim_{\mathrm{WF}} \tau^{\Diamond\Box\Phi}$ iff $\alpha \in L(A^{\Diamond\Box\Phi})$.

For an automaton $A^{\bigvee_i \phi_i}$, define

$$\tau^{\bigvee_i \phi_i}(u) = m^{\exists}(\tau^{\phi_0}(u), \ldots, \tau^{\phi_{|u|}}(u)).$$

Then

$\alpha \in L(A^{\bigvee_i \phi_i})$	iff	(definition of acceptance)
$v_0, v_1, \ldots \models \bigvee_i \phi_i$	iff	(definition of $\bigvee$)
$\exists k : v_0, v_1, \ldots \models \phi_k$	iff	(by induction hypothesis)
$\exists k : \mathrm{WF}\lim_{u \to \alpha} \tau^{\phi_k}(u)$	iff	(by Lemma 14)
$\mathrm{WF}\lim_{u \to \alpha} m^{\exists}(\tau^{\phi_0}(u), \ldots, \tau^{\phi_{\lvert u \rvert}}(u))$	iff	(by definition of $\tau^{\bigvee_i \phi_i}$)
$\mathrm{WF}\lim_{u \to \alpha} \tau^{\bigvee_i \phi_i}(u)$		

The case of an automaton $A^{\bigwedge_i \phi_i}$ is handled similarly, using Lemma 13. □

By Proposition 13, we have:

Corollary 9 Borel sets are analytic and coanalytic.

8.4 Historical Notes

Suslin's Theorem was stated by Suslin in 1917, but the proof was published in [33]; see the introduction of [39]. Kleene's Theorem was proved in [28].

Borel specifications were also studied in [3] where a connection between Borel games and realizability of specifications is established.

Part III

Examples

Chapter 9

Termination Under Fairness Constraints

In this chapter we develop a practical notation and proof method for fair termination.

Unlike previous approaches to this problem [9,15,17,18,20,32,49], our solution involves no program transformations. In Section 9.1 we look at examples of increasing complexity and argue informally how *stack assertions* can explain the behavior of programs under fairness constraints. In Section 9.2 we state the verification condition that stack assertions must fulfill. In Section 9.3 we briefly trace the connection with Rabin measures from Chapter 5.

9.1 Examples

For programs occurring in practice it is usually straightforward to measure termination of a loop. Take for example the program

$$P1: \quad *[\ x < y \ \rightarrow x := x + 1\ ]$$

consisting of loop with a single guarded command, which is executed as long as its guard, $x < y$, is enabled (true). The variables take on integer values. To argue that $P1$ terminates, we use the mapping $\mu_T = \max\{y - x, 0\}$ from program states into the well-founded set of natural numbers $0 < 1 < 2 < \cdots$; here and in the sequel, "T" refers to the hypothesis that the programs terminates. The

mapping μ_T is a progress measure for termination, because its value decreases with each iteration. Thus an infinite computation $p_0, p_1, \ldots$ would produce an infinite descending sequence $\mu_T(p_0) > \mu_T(p_1) > \cdots$, contradicting the well-foundedness of the natural numbers. The progress measure μ_T can be expressed by annotating the program with assertions. For $P1$ a single assertion suffices:

$$\begin{array}{lll} P1': & *[& \big(T : \max\{y - x, 0\}\big) \\ & & x < y \ \rightarrow x := x + 1 \ ] \end{array}$$

Here $\big(T : \max\{y - x, 0\}\big)$ is a *stack assertion*. It asserts that for the termination hypothesis, also called the *T-hypothesis*, the value of the measure μ_T is $\max\{y - x, 0\}$ whenever the loop is to be executed. Thus it could be called a *loop variant*[1] as opposed to a *loop invariant*. The latter expresses a relationship between variables that is preserved under iterations.

With a trivial modification of $P1$, termination is suddenly more intricate. Consider the program

$$\begin{array}{lll} P2: & *[& \ell_a : x < y \ \rightarrow x := x + 1 \ \square \\ & & \ell_b : x < y \ \rightarrow \texttt{skip} \ ] \end{array}$$

where the loop is executed as long as $x < y$ by execution of either of the guarded commands $x < y \ \rightarrow \ x := x + 1$ and $x < y \ \rightarrow$ `skip`; the choice is made nondeterministically. This program will not terminate if the second command is always chosen from some point on. Under assumption of *fair execution*, however, $P2$ always terminates. Here fairness means that if a command is enabled infinitely often, then it is also executed infinitely often. This kind of fairness is sometimes called *strong fairness* [18]. A program *fairly terminates* under a particular fairness constraint if any infinite computation is unfair according to that constraint. For example, $P2$ fairly terminates (under the constraint of strong fairness) because in an infinite computation of $P1$, ℓ_a is only executed finitely often, but enabled infinitely often; thus the computation is unfair with respect to command ℓ_a.

[1]The terms *variant function* or *bound founction* are also used [19].

The key to reasoning about fairness is: if there is no progress towards termination, this can always be attributed to some statement being executed unfairly. For example, when ℓ_b is executed, the T-hypothesis is *dormant*—there is no progress towards termination. Instead, progress towards executing ℓ_a *unfairly* can be measured. To do this we reformulate the stack assertion by appending the hypothesis, called the *a-hypothesis*, that ℓ_a is executed unfairly. Syntactically, this is done by putting an "a" (referring to the label ℓ_a), on top of the underlying T-hypothesis:

$$\left(\frac{a}{T : \max\{y - x, 0\}}\right).$$

The annotated program is now:

$$\begin{array}{lll} P2': & *[& \left(\dfrac{a}{T : \max\{y - x, 0\}}\right) \\ & & \ell_a : x < y \ \rightarrow x := x + 1 \ \square \\ & & \ell_b : x < y \ \rightarrow \ \texttt{skip} \ ] \end{array}$$

Progress is made towards unfair execution in terms of the a-hypothesis whenever ℓ_a is enabled but not executed. Note that for any iteration, either

(a) ℓ_a is enabled and not executed, and the underlying T-measure remains constant (when ℓ_b is executed); or

(T) measure μ_T decreases (when ℓ_a is executed).

We can now argue that $P2$ fairly terminates in terms of (a) and (T) as follows. In an infinite computation, either from some point on (a) always applies, or infinitely often T applies. In the first case, ℓ_a is always enabled but never executed. Hence the computation is unfair with respect to ℓ_a. In the second case, then each time (T) applies, μ_T is decreased, and at the other times, when (a) applies, μ_T is unchanged. This yields an infinite decreasing sequence of natural numbers, a contradiction.

Thus we have proved that for any infinite computation of $P2$, only the first case is possible, i.e. any infinite computation is unfair. Thus $P2$ fairly terminates.

A more complex situation arises if ℓ_a is sometimes not enabled when the ℓ_b command is executed. In this case the stack assertion $\left(\dfrac{a}{T : \max\{y - x, 0\}}\right)$ cannot be applied, because condition (a) is then sometimes not fulfilled. If the program fairly terminates, however, we can employ a progress measure μ_a for the a-hypothesis. For an example, take:

$$
\begin{array}{lll}
P3\colon & *[\ \ell_a : x < y \wedge z \bmod 117 = 0 & \rightarrow x := x + 1\ \ \square \\
& \ell_b : x < y & \rightarrow z := z - 1\ \ ]
\end{array}
$$

This program fairly terminates, because for any infinite computation, ℓ_a can only be executed finitely often and the value of z decreases by one each time ℓ_b is executed; thus ℓ_a is enabled infinitely often. We might annotate the program as follows:

$$
\begin{array}{lll}
P3'\colon & *[\ \left(\dfrac{a : z \bmod 117}{T : \max\{y - x, 0\}}\right) & \\
& \ell_a : x < y \wedge z \bmod 117 = 0 & \rightarrow x := x + 1\ \ \square \\
& \ell_b : x < y & \rightarrow z := z - 1\ \ ]
\end{array}
$$

Here $\mu_a = z \bmod 117$ is a measure of how close $P3$ is to a state in which ℓ_a is enabled. For each iteration of the loop, either

(a') measure μ_T is unchanged, ℓ_a is not executed, and either the value of z was 0 (mod 117) before the execution of ℓ_b, in which case ℓ_a was enabled, or the value of z (mod 117) was between 1 and 116 and decreases by 1; or

(T') measure μ_T decreases.

This ensures that an infinite computation is unfair with respect to ℓ_a. Consider the corresponding infinite sequence of stacks. It must be the case the from some point on, (a') applies to each transition. Thus ℓ_a is only executed finitely often. If from some point on it is never enabled, then μ_a decreases for each iteration thereafter, contradicting the well-foundedness of the natural numbers. Therefore, ℓ_a is enabled infinitely often, and we conclude that any infinite computation is unfair with respect to ℓ_a.

An even more challenging situation occurs when more than one command may be executed unfairly. If we add an empty guarded command to $P3$, we obtain:

$$
\begin{array}{lll}
P4\colon \quad *[\ \ell_a\colon x < y \wedge z \bmod 117 = 0 & \rightarrow x := x + 1 & \square \\
\quad\ \ell_b\colon x < y & \rightarrow z := z - 1 & \square \\
\quad\ \ell_c\colon x < y & \rightarrow \texttt{skip}\ ]
\end{array}
$$

This program fairly terminates, because any infinite computation is unfair with respect to either ℓ_a or ℓ_b. To see this we use the loop variant from $P3'$ modified to explain the lack of progress when ℓ_c is chosen for execution. In that case there is progress neither towards termination nor towards executing ℓ_a unfairly. But there *is* always progress towards something when a program fairly terminates; in fact, when ℓ_c is executed, ℓ_b is a candidate for unfair execution because it is enabled but not executed. Thus we can put the b-hypothesis—that ℓ_b is executed unfairly—on top of T and a. The annotated program then becomes:

$$
\begin{array}{lll}
P4'\colon \quad *[\ \left(\dfrac{b}{\dfrac{a : z \bmod 117}{T : \max\{y - x, 0\}}}\right) & & \\
\quad\ \ell_a\colon x < y \wedge z \bmod 117 = 0 & \rightarrow x := x + 1 & \square \\
\quad\ \ell_b\colon x < y & \rightarrow z := z - 1 & \square \\
\quad\ \ell_c\colon x < y & \rightarrow \texttt{skip}\ ]
\end{array}
$$

This annotation can be used as an argument why $P4$ fairly terminates in a way similar to the previous arguments.

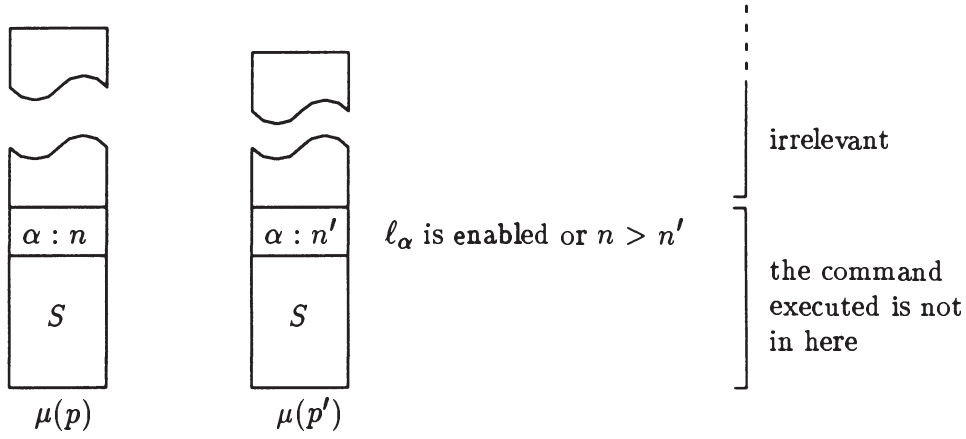
9.2 A Verification Condition For Fairness

In Section 9.1 we developed a notation for reasoning about fairness. For each program we considered an arbitrary infinite computation and argued, using the associated stacks, that the computation was unfair. The rationale for using stack assertions, however, is to avoid reasoning about infinite computations. Therefore we formulate a verification condition for stack assertions. When fulfilled, this condition implies fair termination (under strong fairness); this means that the condition is sound. Completeness holds as well: when a program fairly terminates, it can be annotated with stack assertions such that the verification condition holds.

To fix notation, we define an *α-hypothesis* to be of the form α or $\alpha : n$, where either ℓ_α is a label or α is T, and n is an element of a well-founded ordering. A *stack assertion* $\mu(p)$ is a stack of hypotheses associated with state p with the T-hypothesis at the bottom. The verification condition is expressed in terms of *active* and *dormant* hypotheses; if α denotes a command ℓ_α, then "active" means that progress towards unfair execution of ℓ_α is made, and "dormant" means that ℓ_α is not executed, but there is no current progress towards enabling it. The verification condition can be stated somewhat informally:

(V_F) On any iteration, there is some active hypothesis and the ones below are dormant.

The meaning of this is best explained in terms of a figure:



Here the hypothesis α is at the same position in stacks $\mu(p)$ and $\mu(p')$, and everything below remains unchanged. Two conditions must be fulfilled:

(Act) Hypothesis α is *active*, i.e. either command ℓ_α is enabled (in state p or p') or $n > n'$ holds. Also, ℓ_α is not the command executed in going from p to p'.

(Dor) Every hypothesis β below α is *dormant*, i.e. ℓ_β is not the command executed in going from p to p'. The T-hypothesis is always dormant.

The content of the stacks above α may change in any way.

9.2.1 Examples (continued)

Here is an argument explaining why the stack assertion of $P4'$ satisfies (V_F). Consider an iteration not leading to termination. There are three cases depending on

which command is executed:

ℓ_a: The T-hypothesis is active, because $\mu_T = \max\{y - x, 0\}$ decreases (since $x < y$ holds before ℓ_a is executed). There is nothing beneath the T-hypothesis to check.

ℓ_b: Below the a-hypothesis, the stack remains unchanged. The a-hypothesis is active, because ℓ_a is not executed, and before execution of ℓ_b, either $z = 0 \pmod{117}$ holds, i.e. ℓ_a is enabled, or $z \neq 0 \pmod{117}$ holds, i.e. $\mu_a = z \bmod 117$ decreases.

ℓ_c: The stack is unchanged below the b-hypothesis. The b-hypothesis is active, because ℓ_b is enabled but not executed. The a-hypothesis is dormant, because ℓ_a is not executed.

9.3 Connection with Rabin Measures

The unfair execution of a command ℓ_α can be expressed by the Rabin pair $(\phi_\alpha, \psi_\alpha)$, where ϕ_α states that ℓ_α is enabled and ψ_α states that ℓ_α is executed. The termination hypothesis can be formulated as (ϕ_T, false), where ϕ_T is a predicate expressing that a terminating state is reached (terminating computations are made infinite by repetition of terminating states). Thus fair termination can be expressed as a Rabin condition $\mathcal{C}$.

When the verification condition (V_F) is satisfied for all program transitions, the stack assertions essentially define a Rabin measure of $\mathcal{C}$. Values of the different progress measures in the stack assertion $\mu(p)$ define a path in a pointer tree, and the colors along this path are given by the names of the hypotheses. Yet there are minor technical differences:

- In condition (R) for Rabin measures, satisfaction of an enabling condition ϕ is expressed in terms of the new state. For (V_F), satisfaction of the enabling condition is considered in terms of the old state and the new state.
- A common hypothesis of (V_F) is not necessarily the least common ancestor in the definition of a Rabin measure.
- Two stacks may contain the same pointers, but be colored differently.

- (V_F) does not allow progress to be explained by stacking a new hypothesis. If one translates a Rabin measure into stacks, this situation can be remedied by putting a dummy pointer, greater than the pointers of successors at the same level, in the old stack.

The differences are inconsequential, and the soundness and completeness result of Chapter 5 can easily be adapted to fair termination using the method of stack assertions.

Chapter 10

Termination under Extreme Fairness Constraints

We study a problem whose solution exemplifies a practical approach to handling *extreme fairness*—one of the most general fairness concepts proposed. Extreme fairness arises when a program can execute unfairly in infinitely many ways. As opposed to previous methods [15,35], our approach again shows how to reason about the program without changing it.

10.1 Example

The program $P5$ in Figure 10.1 has infinitely many guarded statements. It is not difficult to see that the program terminates, assuming ℓ_b and ℓ_c are executed fairly (in an infinite fair computation ℓ_b is enabled infinitely often and hence executed infinitely often; thus ℓ_c is enabled infinitely often, whence ℓ_c must be executed, forcing termination. The loop variant $\left(\begin{array}{c} b \\ \hline c \\ \hline T \end{array}\right)$ can be used to show this).

But termination under fairness can also be shown assuming instead that ℓ_c and $\ell_{d_1}, \ell_{d_2}, \ldots$ are executed fairly. We use the loop variant:

$$\left(\begin{array}{c} d_y \\ \hline \cdots \\ \hline d_1 \\ \hline c : y \\ \hline T \end{array}\right)$$

This assertion satisfies the verification condition (V_F). To see this we consider the transitions that do not lead to termination:

ℓ_a: The stack remains unchanged and the d_y hypothesis is active, because it is

$$
\begin{array}{lllll}
P5: & *[\ \ell_a: & y > 0 & \rightarrow \texttt{skip}\ \square & \\
 & \ell_b: & y > 0 & \rightarrow x, y := x+1, x+1\ \square & \\
 & \ell_c: & y = x > 0 & \rightarrow y := 0\ \square & \\
 & \ell_{d_1}: & y = 1 & \rightarrow y := 0\ \square & \\
 & \ell_{d_2}: & y = 2 & \rightarrow y := 1\ \square & \\
 & & & \ldots & \\
 & \ell_{d_k}: & y = k & \rightarrow y := k-1\ \square & \\
 & & & \ldots &]
\end{array}
$$

Figure 10.1: Program with infinitely many guarded commands.

enabled but not executed. Every hypothesis below is dormant, because the command executed, ℓ_a, is not among $c, d_1, \ldots, d_{y-1}$.

ℓ_b: The c-hypothesis is active, because it is enabled but not executed. Note that the value $\mu_c = y$ increases.

ℓ_{d_i}: The c-hypothesis is active, because $\mu_c = y$ decreases and ℓ_c is not executed.

Unfortunately, fair termination is not guaranteed by (V_F) alone. For example take a sequence such as

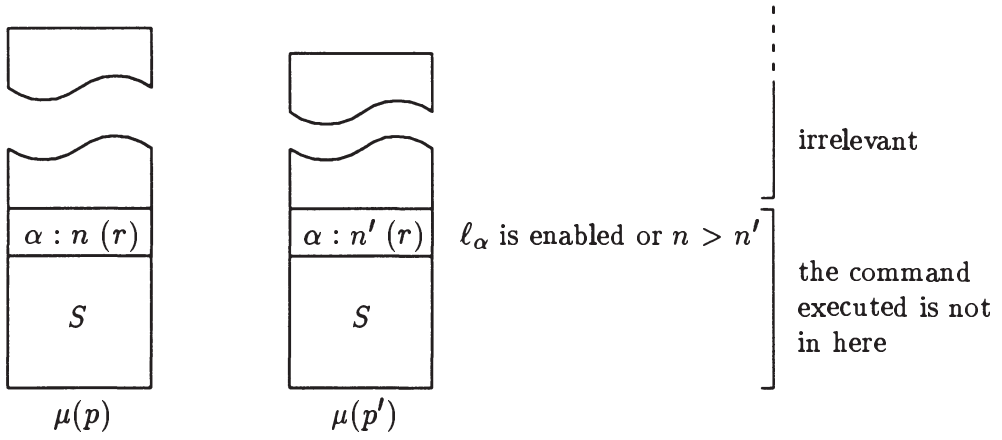
$$\left(\frac{d_1}{T}\right), \left(\frac{\frac{d_2}{d_1}}{T}\right), \left(\frac{\frac{d_3}{\frac{d_2}{d_1}}}{T}\right), \ldots$$

where progress is attributed to first d_1, then d_2, etc. Yet in the limit progress has only been attributed once to every d_i. Thus it might not be the case that any d_i was executed unfairly.

10.2 Verification Condition For Extreme Fairness

To obtain a sound method, we must ensure that for any infinite sequence of stacks, progress happens infinitely often at some level. It suffices to assign a *rank* from a well-founded set to each hypothesis. A hypothesis $\alpha : n$ with rank r is written $\alpha : n\ (r)$. The ranks must be strictly decreasing up through the stack. The verification

condition is the same as before, except that the rank must be unchanged for the active hypothesis and hypotheses below:



This method is sound but not complete, because Rabin measures are not complete for cyclic graphs; see Chapter 6. A complete method can be obtained, however, by letting the measure be a relation that associates a possibly infinite set of stacks to each program state, or by letting the measure be a function of the history of program states.

10.2.1 Example (corrected)

The loop variant for $P5$ can be ranked:

$$\left\lceil \begin{array}{c} d_y\ (1) \\ \hline \cdots \\ \hline d_1\ (y) \\ \hline c : y\ (\omega) \\ \hline T\ (\omega + 1) \end{array} \right\rceil$$

It is not hard to see that this ranking satisfies the extra condition.

ℓ_a: The whole stack remains unchanged.

ℓ_b: The c-hypothesis is active, and the ranking at this level and below remain unchanged.

ℓ_{d_i}: The c-hypothesis is active, and the ranking at this level and below remain unchanged.

Chapter 11

Conclusion

We have introduced the mathematical notion of progress measures and have applied them to problems in program specification and verification. Our methods give a uniform approach to verification problems involving nondeterminism, fairness, and very general liveness properties. Our results establish a link between verification methods and some classical results in descriptive set theory. In particular, we have recast the Suslin-Kleene Theorem as a statement about the relationship between analytic programs and coanalytic specifications.

Our results confirm the following hypothesis in very general circumstances:

> One can verify that the infinite computations of a program satisfy a specification by reasoning only about the program's finite computations.

This approach does not reduce the difficulty of checking program correctness: the problem is Π^1_1-complete. But it does establish that finite computations can be quantified in such a way that progress towards satisfaction of a specification can be measured. Our examples show that in practice this allows us to understand fairness and liveness concepts in terms of local reasoning, but without program transformations.

We hope that our techniques, when expressed in practical programming languages, will help in better understanding distributed algorithms and reactive systems.

Bibliography

[1] M. Abadi and L. Lamport. The existence of refinement mappings. In *Proc. 2. Symp. on Logic in Computer Science.* IEEE, 1988. To appear in *Theoretical Computer Science.*

[2] M. Abadi and L. Lamport. Private communication, November 1989.

[3] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable program specifications. In *Proc. 16th ICALP.* LNCS 372, Springer-Verlag, 1989.

[4] B. Alpern and F.B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, Oct. 1985.

[5] B. Alpern and F.B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2:117–126, 1987.

[6] B. Alpern and F.B. Schneider. Verifying temporal properties without temporal logic. *ACM Transactions on Programming Languages*, 11(1):147–167, January 1989.

[7] American Popular Song. 5 CDs and book, Smithsonian Institution, 1984.

[8] K.R. Apt. Ten years of Hoare's logic: A survey—part I. *ACM Transactions on Programming Languages*, 3(4):431–483, 1981.

[9] K.R. Apt and E.-R. Olderog. Proof rules and transformations dealing with fairness. *Science of Computer Programming*, 3:65–100, 1983.

[10] K.R. Apt, A. Pnueli, and J. Stavi. Fair termination revisited with delay. *Theoretical Computer Science*, 33:65–84, 1984.

[11] A. Arnold. Topological characterizations of infinite behaviors of transition systems. In *Proc. 10th Col. Automata, Languages and Programming*, pages 490–510. LNCS, Vol. 154, Springer-Verlag, 1983.

[12] J.R. Burch, E.M. Clarke, K.L McMillan, D.L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proc. Symp. on Logic in Computer Science*, 1990.

[13] E. M. Clarke, I. A. Draghicescu, and R.P. Kurshan. A unified approach for showing language containment and equivalence between various types of ω-automata. Technical report, Carnegie Mellon University, 1989.

[14] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[15] I. Dayan and D. Harel. Fair termination with cruel schedulers. *Fundamenta Informatica*, 9:1–12, 1986.

[16] R. Floyd. Assigning meaning to programs. In *Mathematical Aspects of Computer Science XIX*, pages 19–32. American Mathematical Society, 1967.

[17] N. Francez and D. Kozen. Generalized fair termination. In *Proc. 11th POPL, Salt Lake City.* ACM, January 1984.

[18] Nissim Francez. *Fairness.* Springer-Verlag, 1986.

[19] David Gries. *The Science Of Programming.* Springer-Verlag, 1981.

[20] O. Grumberg, N. Francez, J.A. Makowsky, and W.P. de Roever. A proof rule for fair termination of guarded commands. *Information and Control*, 66(1/2):83–102, 1985.

[21] D. Harel. Effective transformations on infinite trees with applications to high undecidability, dominos, and fairness. *Journal of the ACM*, 33(1):224–248, 1986.

[22] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, October 1969.

[23] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[24] B. Jonsson. Modular verification of asynchronous networks. In *Proc. Sixth Symp. on the Principles of Distributed Computing*, pages 152–166. ACM, 1987.

[25] N. Klarlund. Verification conditions for ω-automata and applications to fairness. Technical Report TR-1080, Cornell University, February 1990.

[26] N. Klarlund. Verification with disjunctive normal forms. In preparation, 1990.

[27] N. Klarlund and F.B. Schneider. Verifying safety properties using infinite-state automata. Technical Report TR-1036, Cornell University, 1989.

[28] S.C. Kleene. Hierarchies of number theoretic predicates. *Bull. Amer. Math. Soc*, 61:193–213, 1955.

[29] R.P. Kurshan and K. McMillan. A structural induction theorem for processes. In *Principles of Distributed Computing*, pages 239–247, 1989.

[30] L.H. Landweber. Decision problems for ω-automata. *Math. System Theory*, 3:376–384, 1969.

[31] J.v. Leeuwen, editor. *Handbook of Theoretical Computer Science.* North-Holland, 1989. Forthcoming.

[32] D. Lehmann, A. Pnueli, and J. Stavi. Impartiality, justice and fairness: the ethics of concurrent termination. In *Proc. 8th ICALP.* LNCS 115, Springer-Verlag, 1981.

[33] N. Lusin and W. Sierpinski. Sur quelques proprietés des ensembles (A). *Bull. Int. Acad. Sci. Cracovie*, Série A: Sciences Mathématiques:35–48, 1918.

[34] N. Lynch and M. Tuttle. Hierarchical correctness proof for distributed algorithms. In *Proc. Sixth Symp. on the Principles of Distributed Computing*, pages 137–151. ACM, 1987.

[35] M.G. Main. Complete proof rules for strong fairness and strong extreme-fairness. Technical Report CU-CS-447-89, Department of Computer Science, University of Colorado, 1989.

[36] Z. Manna and A. Pnueli. Adequate proof principles for invariance and liveness properties of concurrent programs. *Science of Programming*, 4(3):257–290, 1984.

[37] Z. Manna and A. Pnueli. Specification and verification of concurrent programs by $\forall$-automata. In *Proc. Fourteenth Symp. on the Principles of Programming Languages*, pages 1–12. ACM, 1987.

[38] M. Merritt. Completeness theorems for automata. Technical report, AT&T Bell Laboratories, 1989.

[39] Yiannis N. Moschovakis. *Descriptive Set Theory*, volume 100 of *Studies in Log. and the Found. of Math.* North-Holland, 1980.

[40] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proc. 5th GI conference*, pages 167–183, 1981. In Lecture Notes in Computer Science 104.

[41] A. Pnueli. On the extremely fair treatment of probabilistic algorithms. In *Proceedings 15th ACM Symposium on Theory of Computing*, pages 278–290, 1983.

[42] R. Rinat, N. Francez, and O. Grumberg. Infinite trees, markings and well-foundedness. *Information and Computation*, 79:131–154, 1988.

[43] Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill Book Company, 1967.

[44] S. Safra. On complexity of ω-automata. In *Proc. Foundations of Computer Science.* IEEE, 1988.

[45] A.P. Sistla. On using automata in the verification of concurrent programs. Technical report, Computer and Intelligent Systems Laboratory, GTE Laboratories Inc, 1987.

[46] A.P. Sistla. A complete proof system for proving correctness of nondeterministic safety specifications. Technical report, Computer and Intelligent Systems Laboratory, GTE Laboratories Inc., 1989.

[47] A.P. Sistla. On verifying that a concurrent program satisfies a nondeterministic specification. *Information Processing Letters*, 32(1):17–24, July 1989.

[48] E. Stark. Proving entailment between conceptual state specifications. *Theoretical Computer Science*, 56:135–154, 1988.

[49] F.A. Stomp, W.P. de Roever, and R.T. Gerth. The μ-calculus as an assertion-language for fairness arguments. *Information and Computation*, 82:278–322, 1989.

[50] M. Vardi. Verification of concurrent programs: The automata-theoretic framework. In *Proc. Symp. on Logic in Computer Science.* IEEE, 1987.

[51] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the Conference on Logic in Computer Science*, 1986.

[52] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–99, 1983.

Index

$*$-recursive, *69*
E^*, *6*
$G|U$, *6*
M^*, *6*
M^ω, *6*
Ptr, *40*
$\mathbf{\Sigma^0_1}$, $\mathbf{\Pi^0_1}$,..., *65*
$\perp$, *44*
$\cdot$, *6*
dom f, *6*
rng f, *6*
ℓ-pointer
    of node in pointer tree, *40*
$\exists$-merge of pointer trees, *86*
$\forall$-merge of pointer trees, *86*
$\exists^\infty p(i)$, *6*
$\forall^\infty p(i)$, *6*
$\triangleright$, 3
$\rightarrow$, *24*
$\triangleright_{\text{PR}}$, *28*
$\triangleright_{\text{HI}}$, *30*
$\triangleright_{\text{ND}}$, *31*
$\trianglerighteq$, *40*
$\succeq$, *42*
$\triangleright_{\text{R}}$, *43*
$\trianglerighteq_{\text{Li}}$, *61*
$\trianglerighteq_{\text{WF}}$, *77*
ω, *6*
ω-language, 1
ω^ω, *6*
$\mathcal{P}M$, *6*
$\mathcal{F}M$, *6*
$t[\ell]$, *40*
$t\!\downarrow\lambda$, *40*
$u \propto \alpha$, *6*
$v \rightarrow_U^* v'$, *6*
$\mathcal{R}_A$, *24*
WF T, *76*

alphabet, *24*
ancestor, *40*
    common, *40*
    least common, *41*
assertion
    stack, 90–100
assignment
    color set, *44*
    height, 19, *61*
    Rabin, *40*
    tree, *78*
automaton, 9, 24
    Büchi, 4, *51*
    Borel, *84*
    complete, *24*, 36, 51
    deterministic, 24
    finite-state, 23
    historical, *24*
    Liminf, *71*
    nondeterministic, 83
    program, *23*
    Rabin, 4, *50*
    safety, 3, *24*
    specification, *23*

behavior, 9, *24*

child
    in pointer tree, *40*
color, *40*
    permissible, *44*
    valid, *44*

coloring, *44*
  stable, 44
computation, 9
condition
  Liminf, 19, *61*, 60–74
  Rabin, 4, 14–19, *40*, 39–53
    infinite, 54–59, 64–65
  verification, *2*, 9
    Büchi, 52
    fairness, 95
    history, 31
    Liminf, 72
    ND, 32
    prophecy, 29
    Rabin, 50
    refinement, 27
  WF, 5, *78*

embedding, *77*
event, *24*
execution
  fair, 91

fairness, 90–100
  general, 4
  strong, 91
  weak, 4
formula
  ranked, *84*
function
  approximation, *66*, 76
  bound, 2, 91
  variant, 2, 91

hierarchy
  Borel, 3, *65*
hypothesis
  active, 19, *95*
  dormant, 19, 92, *95*

information
  history, *33*
  prophecy, *35*

language, *24*
Lemma
  History Relation, *30*
  Kleene-Brouwer Ordering, *42*
  Least Common Ancestor, *41*
  Liminf Relation, *61*
  ND Relation, *32*
  Prophecy Relation, *28*
  Rabin Relation, *43*
  WF Relation, 77
level
  of node in pointer tree, *40*
liveness, 14–21

mapping
  possibilities, 26
  prophecy, 26
measure
  ¬Liminf, *73*
  history, 26, *30*
  Liminf, *62*
  Liminf convergence, 68
  ND, 3, 23, *32*
  partial ND, 37
  progress, 3
  prophecy, 26, *29*
  Rabin, 39, *43*, 96–97
  refinement, 9, *27*
  WF, *78*
  WF convergence, 81
method
  complete, 2
  sound, 2

node
  in pointer tree, *40*
nondeterminism, 8–13, 23–38
  bounded, *24*, 36

obeys, *44*
or-merge
  of pointer trees, *81*
ordering
  $\succeq$, 42

Kleene-Brouwer, 42, 72

pair
Rabin, 18, *40*
parent
in pointer tree, *40*
path
eventually χ-permissible, *44*
in pointer tree, *40*
pointer
in pointer tree, *40*
pointer tree, *40*
colored, *42*
finite-path, *40*
height of, *40*
limit, *76*
rankable, 77
prefix
up to level λ, *40*
progress measure, 17
property, *24*
program, 3
safety, 3, 23, *25*

relation
history, *30*
Liminf, *61*
ND, *31*
pointer, *40*
progress, 3, 10
prophecy, *28*
Rabin, *43*
transition, 9, *24*
WF, *77*
run, 9, *24*
partial, 24

set
analytic, *65*, 76, 83
basic open, 65
closed, 25, *65*
coanalytic, *65*, 76
eventually χ-colorable, *44*
history, 30
mono-colorable, *44*
monochromatic, 44
noninvalidating, 48
nonreconfirming, *48*
open, *65*
prophecy, 10, 28
singular, 44
simulation, 37
space
Baire, 25, *65*
state, 24
state, *24*
dead-end, *51*
initial, *24*
invalidating, *40*
program, *23*
reachable, 10, *24*, 37
reconfirming, *40*
specification, *23*
successor
of vertex, *45*
system
transition, 9

transition, 9
tree, *see* pointer tree

uniformly, *70*

variant, *50*, 72
loop, 91
vertex
loop-free, *44*
reachable, *45*