

Copyright © 1998, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

ω -AUTOMATA, GAMES, AND SYNTHESIS

by

Sriram C. Krishnan

Memorandum No. UCB/ERL M98/30

20 May 1998

ω -AUTOMATA, GAMES, AND SYNTHESIS

Copyright © 1998

by

Sriram C. Krishnan

Memorandum No. UCB/ERL M98/30

20 May 1998

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

ω -automata, Games, and Synthesis

by

Sriram C. Krishnan

Doctor of Philosophy in Engineering-Electrical Engineering and Computer Sciences

University of California at Berkeley

Professor Robert K. Brayton, Chair


In this dissertation we investigate the complexity of translation among deterministic ω -automata (DOA), employ our new and improved translations among DOA to derive improved property synthesis algorithms, and consider games to model synthesis problems that arise in practice.

ω -automata are finite state automata that accept infinite strings. The acceptance condition can be one of many types: Buchi, Co-Buchi, Rabin, Streett, etc., each specified by a set of pairs of subsets of states. Restricting the number of pairs permitted in the acceptance condition induces an infinite hierarchy on the languages accepted by DOA. The minimum number of pairs required to accept a language by a DOA—the Rabin Index—is a function of the topological complexity of the language, and determines the structural complexity of any DOA accepting the language. Given a DOA, we address the problem of deciding its Rabin Index; we present a translation of a DOA into an equivalent minimum-pair DOA whose size is exponential in the Rabin Index of the language. We also prove lower bounds to establish the optimality of our translation procedures.

The synthesis of a Finite State Machine (FSM) that satisfies a property specified as a DOA can be viewed as a two-person Gale-Stewart game of perfect information (GSP game) played on the DOA, called the game automaton, between the FSM and its adversarial environment (that supplies the FSM's input). We relate the complexity of synthesis to the complexity of the language describing the property—the game

language; we employ our translations among DOA to derive a synthesis algorithm that decides the winner in a GSP game on a DOA in time that is exponential only in the Rabin Index of the game language (rather than the number of pairs of the game automaton). For instance, we can decide the winner of a GSP game played on a Rabin DOA with n states and h pairs in time $O((nh)^{3k})$, where k is Rabin Index of the game language; this algorithm is optimal. The size of the winner's winning strategy is also at most exponential in the Rabin Index of the game language. We investigate lower bounds for strategy size in relation to the size of the game automaton, and also examine the problem of synthesizing minimum-state strategies.

We study variants of the GSP game on DOA: incomplete information and fair Gale-Stewart games on DOA, as well as GSP games requiring a strategy FSM with no initial state (uninitialized GSP game). These games enable the modeling of synthesis problems arising in practice. All these games are not determined, i.e., there may not be a winner with a deterministic winning strategy. While incomplete information and fair Gale-Stewart games on DOA may be decided by defining appropriate GSP games, this does not appear possible for the uninitialized GSP game. We have devised a synthesis algorithm for the uninitialized GSP game when the game language is a topologically closed set. This yields a synthesis procedure to utilize the maximum flexibility afforded by the safe-replaceability [46] criterion which permits a FSM with no initial state to be replaced by another uninitialized FSM such that the change is not detectable by any environment.



Professor Robert K. Brayton
Dissertation Committee Chair

Acknowledgements

First of all I thank my advisor Prof. Robert K. Brayton for his encouragement, guidance, moral and financial support; the kindness and generosity he has shown towards me; and for the freedom he has given me. I am also thankful to Profs. Pravin P. Varaiya and Leo A. Harrington for kindly serving on my qualifying exam and dissertation committees, and Prof. Kathy Yelick for serving on my qualifying exam committee.

It has been a pleasure to collaborate closely with Anuj Puri. I had many interesting discussions with Anuj on ω -automata and games. Many of the results in Chapter 2 of this thesis were obtained working jointly with Anuj; we've also discussed the subject matter of Chapters 3 & 4. Anuj's flair for clarity has had a positive influence on me.

I have also had the opportunity to conduct joint research with Amit Narayan, Rajeev Murgai, Ramin Hojati, and Sunil "Linus" Khatri. I was involved in the HSIS project am thankful to all the members of the HSIS team for the opportunity to work with them. I have had useful discussions with Ken McMillan. The problem addressed in Chapter 7 of this thesis came up during Vigyan Singhal's talk at the NATO synthesis workshop held in Feb. 1998 at the Cadence Berkeley Labs. I prepared for the preliminary exam along with Henry Sheng, Ramin Hojati, and Tom Shiple.

I have sought the counsel of my more experienced colleagues on various matters at various times. I would like to thank Desmond Kirkpatrick, Narasimha Bhat, Rajeev Murgai, Sunil Khatri, and Tiziano Villa for their thoughtful advice.

An enriching aspect of the Berkeley experience for me has been the people I have met here, interacted with, and made friends with. The CAD group has been a great and open research environment made up of bright and helpful people, and it has been a great pleasure and privilege to be part of it. I also enjoyed the IHouse experience—living there first and eating dinner there the last couple of years. Visiting the Vedanta Society has also been part of my life. Here is a partial list (in random order, put down as I recall them from my RAM) of the people whose association and friendship I am happy to acknowledge: Adnan Aziz, Felice Balarin, Mark Beardslee, Narasimha & Sashikala Bhat, Timothy Kam, Luciano Lavagno, Alex Saldanha, Rick

McGeer, Rajeev & Neelakshi Murgai, Wendell Baker, Anuj Puri, Premal & Tanvi Buch, Luca Carloni, Edoardo & Tokiko Charbon, ST Cheng, Phil Chong, Stephen Edwards, Anuj Puri, Eric Felt, Chris Lennard, Narendra Shenoy, KJ Singh, Wilsin & Pei-lin Gosti, Gitanjali & Sanjay Sarma, Harry Hsieh, Huey-yih Wang, Tom & Suzanne Shiple, Jagesh & Alpa Sanghavi, Adrian Isles, Sunil Khatri, Amit Narayan, Amit Mehrotra, Gurmeet Singh Manku, Mukul Prasad, Lixin Su, Subarna Sinha, E. Mahalingam, Anu Bhat, Desmond Kirkpatrick, Ramin Hojati, Yuji Kukimoto, Yosi-nori Watanabe, Freddy Mang, Shaz Qadeer, Orna Kupferman, Sriram Rajamani, Andreas Kuehlmann, Praveen & Jyothi Murthy, Sriram Sundararajan, Kumud Sanwal, Rajeev Ranjan & Ms. RenU, Vigyan & Sonali Singhal, Marco Sgroi, Serdar Tasiran, Bassam Tabbara, Iasson Vassilou, Ken Yamaguchi, Kiran, Arnab Nilim, PR Hariharan, Varghese George, the Passerone brothers, Ravi Gunturi, Ski, et. al.

Thanks are due to Brad Krebs, Mike Kiernan, and Judd Reiffin for maintaining an excellent computing environment. Kia Cooper and Flora Oviedo have been very helpful with reimbursements, supplies, etc.. Mary Byrnes, Heather Brown, and Ruth Gjerde in the graduate office have been invaluable in managing the forms required by Sproul, and have been most helpful and courteous whenever I went to their office with a problem.

The following funding sources paid for my stay in Grad School: California Microelectronics Fellowship, California MICRO program, AT & T Bell Laboratories, Fujitsu Laboratories of America, and the Semiconductor Research Consortium (SRC).

Contents

List of Figures	viii
1 Introduction	1
2 Structural Complexity of ω-automata	7
2.1 Introduction	7
2.2 Background & Notation	9
2.2.1 ω -automata	12
2.3 Topology on Σ^ω	19
2.4 Open and Closed ω -regular languages	20
2.5 G_δ languages—DBA-realizable languages	22
2.6 Lower bounds for translating into DBA	27
2.7 Testing for DBA-realizability	29
2.7.1 Checking for Buchi-typeness	29
2.8 F_σ -languages—Co-DBA realizable languages	32
2.8.1 $G_\delta \cap F_\sigma$ regular languages	32
2.9 $G_{\delta\sigma} \cap F_{\sigma\delta}$ — ω -regular languages	32
2.10 Translating to a minimum-pair automaton	37
2.10.1 Excluding <i>supLan</i>	39
2.10.2 DRA to DRCA construction	41
2.11 Characterizing deterministic Chain automata	46
2.12 Computational complexity of determining RI/SI	47
2.12.1 DMA	47
2.12.2 DSA & DRA	47
2.12.3 Determining the Streett/Rabin Index is NP-hard	48
2.13 Lower bounds for translation in $G_{\delta\sigma} \cap F_{\sigma\delta}$	50
2.13.1 Lower bounds for translating DSA & DRA	51
2.13.2 Lower bound for translating DMA	54
2.14 Minimization of DOA	56
2.15 Determinization	59

2.16	Minimum witnesses for non-empty ω -automata	61
2.17	Summary	63
3	Two-person Games of Perfect Information on ω-automata	65
3.1	Introduction	65
3.2	Game on ω -automaton	66
3.2.1	Players' strategies	67
3.3	McNaughton's graph game	72
3.3.1	Players' strategies	74
3.3.2	Translating ω -automaton games into graph games	75
3.4	Open and Closed games	77
3.5	Buchi and Co-Buchi games	79
3.6	1-pair Rabin and Streett games	81
3.6.1	Emerson-Jutla's algorithm	83
3.7	General Rabin and Streett games	84
3.7.1	Emerson-Jutla's algorithm	88
3.7.2	Deciding Chain games	89
3.7.3	Deciding a Rabin game in "Rabin Index time"	90
3.7.4	Putting it in perspective	94
3.8	Games and tree automata	95
3.9	Lower bounds for memory and strategy-size	99
3.9.1	A lower bound for the General Muller game	103
3.9.2	Lower bound for general Streett games	104
3.9.3	Relationship between automata bounds and game bounds	108
3.10	Summary	109
4	Two-person Games of Incomplete Information on ω-automata	110
4.1	Introduction	110
4.2	Strategies for playing	111
4.3	Deciding GSI games on ω -automata	114
4.3.1	Bounds	117
4.4	Applications of GSI games	118
4.4.1	Watanabe-Brayton FSM-component synthesis problem	118
4.4.2	Supervisory control of FSMs	120
4.5	Summary	121
5	Fair games on ω-automata; a synthesis example	122
5.1	Introduction	122
5.2	Fair games	123
5.2.1	Deciding FGSP games	123
5.2.2	(Non-)Determinacy of FGSP games	124

5.3	Example	126
5.4	Summary	136
6	Minimum State Strategies	137
6.1	Introduction	137
6.2	Computational complexity of deriving minimum-sized strategies	138
6.3	Insufficiency of the enumeration of ML-strategies	143
6.4	FSMs on the state-subset space	145
6.4.1	Finding the smallest strategy FSM for closed winning condition	146
6.5	Computing the smallest memory-based winning strategy	148
6.6	Computing the smallest strategy FSM for non-closed winning conditions	150
6.7	Summary	152
7	Synthesizing uninitialized strategies	154
7.1	Introduction	154
7.2	Uninitialized strategies	155
7.3	Uninitialized Subset FSM	155
7.4	Synthesizing uninitialized strategies for closed games	157
7.4.1	E-machine for uninitialized strategies for player-0 in closed games	158
7.4.2	Uninitialized strategies for player-1 in open games	158
7.5	Synthesis for safe replaceability	159
7.6	Non-closed games	159
7.7	Determinacy	160
7.8	Summary	161
8	Conclusion	163
	Bibliography	167

List of Figures

2.1	Finite state transition structure	11
2.2	Hasse Diagram of SCSs of an automaton	18
2.3	Borel hierarchy in Cantor space	20
2.4	$supLan$ not an invariant of the language	23
2.5	DMA not Buchi-type but DBA-realizable	25
2.6	Positive chain of index 2 in a 2-pair DRA	33
2.7	DRA2DRCA in action: effect on polarity of SCSs	38
2.8	DSA \mathcal{A}_n accepting L_n	53
2.9	Counter-examples to DBA minimization possibilities	57
3.1	Simple Buchi Game	67
3.2	Mealy winning strategy for player-1 starting from state A in game of Figure 3.1	70
3.3	Moore winning strategy for player-0 starting from state C in game of Figure 3.1	70
3.4	A Game Graph	73
3.5	Memory-less winning strategy for player-0 in game of Fig. 3.4	76
3.6	The partition of winning nodes in a Buchi game	80
3.7	A Game where player-0 wins but only with memory	84
3.8	A Rabin game where 2 pairs <i>together</i> determine a win for player-0	85
3.9	Streett game where player-0 needs 2^n memory	106
4.1	GSI game without a winner	113
4.2	The Watanabe-Brayton component-FSM synthesis problem	119
4.3	The FSM supervisory control problem	120
5.1	The game language and fairness constraints: containment relationship	124
5.2	A fair GSP game with no winner	125
5.3	Architecture of the traffic light controller	127
5.4	The timer	128
5.5	HLC: Highway traffic light controller FSM	128
5.6	FLC: Farm road traffic light controller FSM	129

5.7	Safety: HL and FL not green simultaneously	130
5.8	Justice: If car present and timer expired, eventually FL is green . . .	130
5.9	Recurrence: HL green infinitely often	130
5.10	Composite property P	132
5.11	Product machine of timer and farm-road control modules	133
5.12	Highway Light Control Module I: HLCI	134
5.13	Highway Light Control Module II: HLCII	135
5.14	Highway Light Control Module III: HLCIII	135
6.1	SMLS instance for 3-CNF instance $(x_1 + x_2 + x_3)(\neg x_1 + x_4 + \neg x_3)(\neg x_4 + \neg x_2 + \neg x_3)$	139
6.2	Transformation of k -colorability instance (a) to MSFSM instance (b)	142
6.3	A game automaton where player-0 needs memory to yield a minimum-state strategy FSM	144
7.1	Closed game where neither player has an uninitialized winning strategy	161

Chapter 1

Introduction

Today's Very Large Scale Integrated (VLSI) circuit designs are conceived of as a set of interacting Finite State Machines (FSMs) and described in a hardware description language (HDL) such as Verilog. The specification or requirement of the design is for the most part stated informally. Validation, or verification that the design behaves as intended, is performed by simulating the design with a set of input stimuli (vectors) and checking that the behavior of the design meets the specification. Validation is essentially an ex-post-facto act, engaged in after the design effort. Once a design has been validated or verified it is optimized and fabricated on silicon.

The increasing complexity of designs has found the present approaches to obtain a validated design wanting. Recently, formal verification, verification methods involving proving that a formal mathematical model of the design conforms to a mathematical specification, has received much attention. A related, promising, and different approach is formal property synthesis—the synthesis of a design from the properties required of the design (thereby rendering verification unnecessary).

The problems addressed in this thesis relate to formal property synthesis. In particular we treat the case that the property required of the interaction between the design and its environment is specified as an ω -automaton. ω -automata are finite state automata that accept infinite sequences, and were historically first introduced by Buchi [6] to decide the monadic second-order theory of one successor (S1S). ω -automata enable both the modeling of the design at a high level of abstraction, as well

as the specification of properties required of the design. For instance, ω -automata enable us to specify that a resource such as a bus is eventually granted upon request without specifying an explicit deadline. Furthermore, since computing systems maintain an ongoing nonterminating interaction with their environment, ω -automata provide a natural mathematical tool for their modeling. Amongst formal models, ω -automata are closer to the heart of computer hardware designers, being simple generalizations of finite state machines (FSMs). For ω -automata, acceptance is defined by Boolean formulae of infinitely repeating distinguished states; different commonly used Boolean formulae have given rise to the different ω -automata in the literature: Buchi, Co-Buchi, Rabin, Streett, Chain, Muller, etc. These Boolean formulae are given as a set of pairs of subsets of states.

When ω -automata are employed as the modeling tool, the verification problem, amounts to checking that the language of (the set of behaviors of) the design is contained in (admitted by) the language of the property. Verification using ω -automata is efficiently solvable [16] and there are commercial tools available [28].

The sequential synthesis problem is: Given an ω -automaton, the property, specifying the proper interactions between the sequential circuit to be designed and its environment, is there a sequential circuit, namely a FSM, admitted by the property. This problem has come to be known as Church's problem [9].

Sequential synthesis of a property is not mere optimization like combinational logic synthesis. A property for a combinational circuit is a Boolean relation $R \subseteq \{0, 1\}^n \times \{0, 1\}^m$. Since there always is a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ satisfying the relation R , i.e., such that for every $x \in \{0, 1\}^n$, if there exists $y \in \{0, 1\}^m$ such that $(x, y) \in R$, then $(x, f(x)) \in R$ [4], the combinational circuit synthesis problem amounts to picking a function subject to optimization objectives. Whereas, for a sequential circuit, a property relation $R \subseteq I^\omega \times O^\omega$, given as an ω -automaton, indicates the allowed output sequences for input sequences. A FSM that satisfies the property is essentially a function $f : I^* \rightarrow O$ such that for all $i_0 i_1 i_2 i_3 \dots \in I^\omega$, $(i_0, f(i_0)), (i_1, f(i_0 i_1)), \dots, (i_j, f(i_0 i_1 \dots i_j)) \dots \in R$. It is no longer true that every property is feasible, i.e., there is not always a FSM that satisfies the property. For instance, it is easy to see that the relation $R \subseteq \{0, 1\}^\omega \times \{a, b\}^\omega$ with $R(01\{0, 1\}^\omega, a^\omega)$

and $R(00\{0,1\}^\omega, b^\omega)$ is not feasible. Therefore sequential synthesis involves checking for feasibility and if feasible, synthesizing an “optimal” circuit.

Church’s problem or sequential synthesis is significantly more complicated than combinational synthesis. Church’s problem was solved by Buchi and Landweber in 1969 [7]. Following McNaughton, it is pointed out in [7] that a FSM that satisfies the property is a winning finite state strategy in a two-person Gale-Stewart game played on an ω -automaton. Viewing the synthesis problem as a game provides a convenient and natural metaphor for the adversarial interaction between the environment of the module to be synthesized and the module. In addition, we show that it leads to efficient algorithms for synthesis and have developed variations of the Gale-Stewart game to model and solve other practical synthesis problems (that cannot be posed as Church’s problem). The flexibility afforded for synthesis will be the set of winning strategies for the module to be synthesized in an appropriate game, and if the property is feasible, i.e., if the module wins the game, the objective would be to pick out a winning strategy subject to optimization objectives.

Translation between different deterministic ω -automata (DOA) has an intimate relation to deciding the feasibility and synthesis of properties specified as ω -automata. In this dissertation we present optimum translations between deterministic ω -automata, and utilize these to obtain new improved decision procedures for Church’s problem. We also study variants of the two-person Gale Stewart game on an ω -automaton to model practical synthesis problems, and present synthesis algorithms for them.

Chapter 2 investigates the structural complexity of ω -automata. The structural complexity of a deterministic ω -automaton is determined by the transition structure of the automaton and the number of pairs used to specify the acceptance condition, and this is also related to the topological complexity of the language. Given a deterministic ω -automaton, we consider the problem of determining the structural complexity of the automaton, and translate the automaton into an equivalent automaton with a minimum number—the Rabin Index number—of pairs. Our new translations are exponential in the Rabin Index rather than the number of pairs, and are asymptotically no worse than previous constructions [45]; we prove lower bounds showing the worst-case optimality of our translations. We also enquire if there is a

determinization of ω -automata whose complexity is a function of the Rabin Index of the language.

Our results build on the structural characterization of the minimum number of pairs provided by Landweber [29], Wagner [55], and Kaminski [24], to derive algorithms to effect optimal translations between deterministic ω -automata. These results add to Safra's [45] to give us a better picture of the relative succinctness and complexity of the different deterministic ω -automata, within the subclasses of the ω -regular languages arising from restricting the number of pairs allowed.

In Chapter 3 we consider two-person Gale-Stewart games played on deterministic ω -automata, and problems of deciding the winner and synthesizing the winner's strategy; these games are equivalent to Church's problem. The time to decide the winner and the size of the winner's strategy vary quite widely depending on the type, whether Buchi, Rabin, etc., of automaton the game is played on—the game automaton. We employ the translations from Chapter 2 to derive synthesis algorithms that run in time exponential in the Rabin Index of the game automaton, and synthesize winning strategies whose size is at most exponential in the Rabin Index. Previous algorithms [38, 14] were exponential in the number of pairs; ours is better in general, and asymptotically no worse. In terms of time complexity previous results [14] rule out an improved procedure. We also attempt to prove lower bounds on the strategy size as a function of the size of the game automaton, as a means of ruling out the existence of a synthesis algorithm that can produce smaller winning strategies. Gale-Stewart games can also model tree automata [40]; our results imply improved decision procedures for testing nonemptiness of tree automata, and hence imply improved decision procedures for testing satisfiability of branching time temporal logics [14].

Small winning strategies translate to smaller VLSI circuits. In Chapter 6 we address the synthesis of minimum-state strategies. We prove computational complexity lower bounds as well as present exact algorithms for the problem.

The game corresponding to Church's problem is a Gale-Stewart game of perfect information on a deterministic ω -automaton. Property synthesis problems arising in practice, such as to synthesize a component that interacts with a set of previously

synthesized components, do not always fit this paradigm. We propose and investigate variants of the perfect information Gale-Stewart game to model these synthesis problems.

Chapter 5 examines two-person Gale-Stewart games of incomplete information on deterministic ω -automata. Here, each player may only observe, and has to base strategic decisions on, the observable part of the opponent's actions. The winner in this game can be decided, and his winning strategy derived, by considering an appropriate Gale-Stewart perfect information game, but at a higher computational cost. These games are not determined, in that some games do not have a winner with a deterministic winning strategy.

Fairness constraints aid in more accurately modeling the behavior of modules upon abstraction. In Chapter 6, we study fair Gale-Stewart games on deterministic ω -automata. These games can again be decided by defining appropriate perfect information games. If neither player's fairness constraint is a topologically closed language, these games may not have a winner. We also consider a small illustrative example.

Recently [46, 47] the design of uninitialized FSMs—sequential circuits without specified initial states—has become important. Given a property specified as a deterministic ω -automaton, we address, in Chapter 7, the problem of synthesizing an uninitialized FSM satisfying the property. We consider an appropriate game and solve the case that the language of the game automaton is a topologically closed set. This also yields a method to exploit the complete flexibility for safe-replaceability [46]: the problem of replacing an uninitialized FSM by another uninitialized FSM such that no environment is able to detect the change. Unlike incomplete information or fair games, we have not been able to transform the game appropriate to the problem of synthesizing an uninitialized FSM to a Gale-Stewart perfect information game on an ω -automaton. These games (even closed games) are not determined.

We conclude in Chapter 8 by summarizing our results and indicating directions for ongoing and future work.

Chapters 2 and 3 set out the definitions of ω -automata and games of perfect information and should be read before the subsequent chapters. Chapters 4 through

7 are fairly independent and may be read in any order.

The following are the main points of the report:

The first part of the report discusses the background of the project and the objectives of the study. It also outlines the methodology used in the research and the scope of the study.

The second part of the report presents the results of the study. This includes a detailed analysis of the data collected and the findings of the research. The results are presented in a clear and concise manner, with appropriate use of tables and figures to illustrate the data.

The third part of the report discusses the implications of the findings and the conclusions drawn from the study. It also provides recommendations for further research and practical applications of the findings.

The final part of the report is a summary of the key findings and conclusions. It also includes a list of references and an appendix containing additional data and information.

Chapter 2

Structural Complexity of ω -automata

2.1 Introduction

ω -automata are finite state automata that accept infinite sequences. The run of an infinite sequence on a finite state transition structure visits some set of states infinitely often, call the infinity set. The sequence is accepted provided it satisfies the acceptance condition, which is a Boolean formula over Boolean variables, one for each state. The acceptance condition is satisfied provided the truth assignment that makes variables in the infinity set true is a satisfying assignment for the Boolean formula defining the acceptance condition.

Restricting the syntax of the Boolean formulae used to define the acceptance condition, gives rise to the different ω -automata in the literature—Buchi, Co-Buchi, Rabin, Streett, Chain, and Muller. The Muller acceptance condition is the minterm canonical form of the acceptance condition, and is hence the most verbose. The Buchi and Co-Buchi acceptance conditions are the simplest, and hence the most succinct. The Buchi condition is defined by a disjunctive formula, and the Co-Buchi conditions by the negation of a disjunctive formula. Particular forms of Boolean combinations of Buchi and Co-Buchi conditions define the Rabin and Streett acceptance conditions.

Deterministic ω -automata have varying degrees of expressiveness, depending on

the form and size of the acceptance conditions used to define them. The size, refers to, for instance the number of disjuncts or pairs in the Rabin condition. Deterministic Buchi and Co-Buchi automata can express a strict subclass of the ω -regular languages. Restricting the number of pairs in deterministic Rabin and Streett automata induces a strict hierarchy of languages within the ω -regular languages.

Given a deterministic ω -automaton, we consider the question of translating the acceptance condition, i.e., constructing an equivalent ω -automaton with a different acceptance condition. This often entails a change in the transition structure. The different acceptance conditions vary quite widely in their succinctness; an acceptance condition may be able to express an ω -regular language exponentially more succinctly than another acceptance condition. We are interested in effecting the translation efficiently, minimizing the size of the acceptance condition and the transition structure. This problem may be viewed as a logic minimization problem of sorts, with the added factor of the transition structure.

The minimal form of the acceptance condition that a given deterministic ω -automaton can admit is a function of the form and size of the formula used to describe its acceptance condition, as well as the transition structure of the automaton. The interaction between the two—the structural complexity of the ω -automaton—influences characteristics of the automaton that determine the location of the language in the hierarchy of ω -regular languages, and the minimal form of the acceptance condition that may be used to define the language as a deterministic ω -automaton. The chain condition is a special form of the Rabin and Streett acceptance conditions that admits an easy minimization of its formula. The hierarchy on the ω -regular languages also has a strong connection to the topological properties of languages [55, 50], as well as the descriptive complexity of properties of systems [31].

We provide essentially optimum translations between the acceptance conditions, as well as minimizations of the acceptance conditions. The complexity of our translations is a function of the location of the language in the hierarchy on the ω -regular languages. Our results add to Safra's [42, 44, 45] results in understanding the complexity of ω -automata. Our results show that the relative succinctness of the different acceptance conditions gets amplified higher in the hierarchy. We also consider decision

problems relating to locating a given language in the hierarchy.

ω -automata are used to construct qualitative abstractions of reactive systems to be modeled for analysis, and stating the properties that the systems satisfy [31]. The results in this chapter are of interest from an automata-theoretic perspective, with important applications in the area of synthesis, the subject of subsequent chapters of this thesis.

This chapter is structured as follows. Section 2.2 presents the notation and an introduction to ω -automata. In Section 2.3 we describe the topology relevant to ω -regular languages, and in Section 2.4 we consider translation of open and closed languages. Sections 2.5 through 2.8 deal with the class of languages realizable by deterministic Buchi automata. Sections 2.9 through 2.13 deal with translation in the (full) class of ω -regular languages. In Section 2.14 we consider the problem of minimizing the number of states in a deterministic ω -automaton. In Section 2.15 we investigate if the complexity of determinization is a function of the structural complexity of the resulting deterministic automaton. Section 2.16 deals with minimally certifying the nonemptiness of an ω -automaton. We summarize the results of this chapter in Section 2.17.

2.2 Background & Notation

An *alphabet* is a finite non-empty set of *letters*. For example, alphabet $\Sigma = \{a, b, c\}$, consists of three letters a , b , and c . Let ω denote the set of natural numbers $\{0, 1, 2, 3, \dots\}$. For $n \in \omega$ let \bar{n} be the set $\{0, 1, 2, \dots, n-1\}$. A *word* or *string* is a juxtaposition or concatenation of letters. A finite word over alphabet Σ , or $*$ - Σ -word, of length n is a concatenation of n letters, i.e., a function from $\bar{n} \rightarrow \Sigma$. An ω -word, or ω -sequence, over alphabet Σ or ω - Σ -word is an infinite concatenation of letters from Σ , i.e., a function from ω to Σ , and we say that the word has length ω . For an alphabet Σ , Σ^* is the set of all finite words, Σ^ω the set of all infinite words, and $\Sigma^\dagger = \Sigma^* \cup \Sigma^\omega$. For $x \in \Sigma^\dagger$ let $|x|$ denote the length of the word, and $x[i]$ denote the i^{th} letter in the word, $i < |x|$. If $\Sigma = \Sigma_1 \times \Sigma_2$, and $s = (s_1, s_2) \in \Sigma$, the *projection* of s to Σ_1 , $\Pi_{\Sigma_1}(s)$, is defined to be s_1 . For $x = (x_{0_1}, x_{0_2})(x_{1_1}, x_{1_2}) \cdots \in (\Sigma_1 \times \Sigma_2)^\dagger$,

$\Pi_{\Sigma_1}(x) = x_{0_1}x_{1_1}\dots$. For $x, y \in \Sigma^\dagger$, the concatenation of x and y , denoted xy , is xy if $x \in \Sigma^*$, and x otherwise; we say x is a *prefix* of y denoted $x \leq y$ if $y = xz$ for $z \in \Sigma^\dagger$, and x is a *strict prefix* of y , denoted $x < y$ if $x \leq y$ and $x \neq y$. $L \subseteq \Sigma^*$ is called a **-language* and $L \subseteq \Sigma^\omega$ is called an ω -language. The concatenation of a **-language* L_* and an ω -language L_ω , is denoted L_*L_ω , and is $\{xy \mid x \in L_* \text{ and } y \in L_\omega\}$. Given a language $L \subseteq (\Sigma_1 \times \Sigma_2)^\omega$, the projection of L to Σ_1 , $\Pi_{\Sigma_1}(L)$ is $\{\sigma_1^0\sigma_1^1\dots \in \Sigma_1^\omega \mid \exists \sigma_2^0\sigma_2^1\dots \in \Sigma_2^\omega \text{ such that } (\sigma_1^0, \sigma_2^0)(\sigma_1^1, \sigma_2^1)(\sigma_1^2, \sigma_2^2)\dots \in L\}$.

A traditional finite automaton [41, 22], or **-automaton*, is a quintuple $\mathcal{A} = (Q, q_0, \Sigma, \delta, F)$, where:

- Q , is a finite set of *states*, also called the state-set or state space, or simply states of the automaton
- $q_0 \in Q$, is the *initial state*
- Σ , is a finite alphabet
- $\delta \subseteq Q \times \Sigma \times Q$, is the *transition relation*
- $F \subseteq Q$, is the set of *final states*

The first four components (Q, q_0, Σ, δ) comprise the *transition structure*. The transition structure defines a directed graph, the *State Transition Graph* (STG), on vertex set Q , with Σ -labeled edges corresponding to elements of δ . Figure 2.1, shows a transition structure on state-set $\{A, B\}$, initial state A (indicated by an arrow), alphabet $\{a, b\}$, and transition relation depicted by the edges of the STG.

A $*\text{-}\Sigma$ -word σ has a *run* $r_\sigma \in Q^*$, provided $r_\sigma[0] = q_0$, $(r_\sigma[i], \sigma[i], r_\sigma[i+1]) \in \delta$, and $|r_\sigma| = |\sigma| + 1$. The word $aabba$ has the run $AAABBA$ on the transition structure of Figure 2.1. If for some state $q \in Q$, and letter $x \in \Sigma$, a transition is not defined, i.e., $\forall q' \in Q, (q, x, q') \notin \delta$, the transition structure is said to be *incomplete*, and otherwise it is *complete*. If δ could be equivalently expressed as a function from $Q \times \Sigma \rightarrow Q$, then we say the transition structure is *deterministic*, and *nondeterministic* otherwise.

$\delta : Q \times \Sigma \rightarrow Q$ is naturally extended to $\delta : Q \times \Sigma^+ \rightarrow Q$ as follows: for $c \in \Sigma^+$,

$$\delta(q, c) = \begin{cases} \delta(q, c[0]) & \text{if } |c| = 1 \\ \delta(\delta(q, c[0]), c[1] \dots c[|c| - 1]) & \text{otherwise.} \end{cases}$$

Every word has a unique run on a complete deterministic transition structure. We say a state $q \in Q$ is *reachable*, if for some $*\Sigma$ -word σ , some run r_σ contains q , i.e., $r_\sigma[i] = q$ for $i \leq |\sigma|$; in this case we also say the run r_σ *reaches* or *visits* q . Run r is *accepting* if $r[|r| - 1] \in F$, i.e., if the run ends in a final state.

The $*\Sigma$ -word σ is said to be accepted provided it has an accepting run r_σ . The language, of, accepted by, $*\Sigma$ -automaton \mathcal{A} , denoted $L_*(\mathcal{A})$, is the set of all accepted words. A language $L \subseteq \Sigma^*$ which is the language accepted by a $*\Sigma$ -automaton is called a **-regular* language. The language of a $*\Sigma$ -automaton, $\mathcal{A} = (Q, q_0, \Sigma, \delta, F)$ is empty if no final state is reachable. Every $*\Sigma$ -regular language is in fact accepted by a deterministic $*\Sigma$ -automaton. Also, $*\Sigma$ -regular languages are closed under union, intersection, complementation, and projection [22].

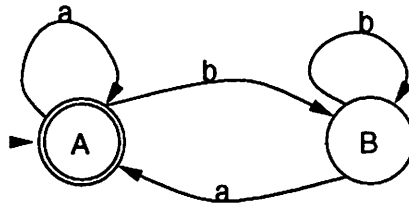


Figure 2.1: Finite state transition structure

We say an automaton is (non-)deterministic, (in-)complete if its transition structure is respectively (non-)deterministic, (in-)complete. Figure 2.1 shows a simple complete deterministic $*\Sigma$ -automaton. There is only one final state, the state labeled A . The $*\Sigma$ -regular language accepted by the $*\Sigma$ -automaton is all finite strings that end in a , and is represented by the regular expression $(a + b)^*a$. We use the acronym DFA for deterministic $*\Sigma$ -automaton, and NFA for nondeterministic $*\Sigma$ -automaton.

2.2.1 ω -automata

An ω -automaton [52] is $\mathcal{A} = \langle T, \phi \rangle$, where T is a transition structure and ϕ is the acceptance condition. The transition structure is $T = \langle Q, q_0, \Sigma, \delta \rangle$, analogous to $*$ -automata, as are the notions of deterministic, nondeterministic, and complete. We use the acronym DOA for a deterministic and complete ω -automaton, and NOA for a non-deterministic ω -automaton. The run for an ω -word σ , r_σ is defined similar to the case of $*$ -automata, but it is now an ω - Q -word.

Since Q is a finite set, the run of an ω -word will necessarily visit some states infinitely often. The *infinity* set of a sequence ψ over a finite set Z , i.e., $\psi \in Z^\omega$, denoted $\text{inf}(\psi)$, is the set of elements in Z that occur infinitely many times in ψ , i.e., $\text{inf}(\psi) = \{z \mid z \in Z \text{ and } |\{i \mid i \geq 0 \text{ and } \psi[i] = z\}| = \infty\}$. The infinity set of the run r_σ , $\text{inf}(r_\sigma)$, is thus the set of states that are visited infinitely often by r_σ . Beyond some index I , all the states visited by r_σ will be from $\text{inf}(r_\sigma)$. And, for any $i \geq I$, and any states $q_1, q_2 \in \text{inf}(r_\sigma)$, there exist indices $l \geq k \geq i$, such that $r_\sigma[k] = q_1$ and $r_\sigma[l] = q_2$. Thus, the subgraph of the STG T on the vertex set $\text{inf}(r_\sigma)$ is *strongly connected*, since for every $q_1, q_2 \in \text{inf}(r_\sigma)$ there is a directed path (along r_σ) from q_1 to q_2 ; we say $\text{inf}(r_\sigma) \subseteq Q$ is a *Strongly Connected Subset* (SCS) (of T). A maximal SCS is called a *Strongly Connected Component* (SCC); for every $q \in Q$, the SCC q is contained in, $\text{SCC}(q)$ is unique. $S \subseteq Q$ is said to be a *non-trivial* SCS if the subgraph of T on the vertex set S is a SCS with at least one edge. For every non-trivial SCS $S \subseteq Q$, with at least one reachable state, for some word σ , $\text{inf}(r_\sigma) = S$.

The *acceptance condition* ϕ , the condition to be satisfied by an accepting run, is a Boolean formula, where the Boolean variables are the states $Q = \{q_1, \dots, q_n\}$.

Definition 2.2.1 *A Boolean formula is generated by the following rules*

- 1) $q_i \in Q$ is a Boolean formula
- 2) If ϕ_1, ϕ_2 are Boolean formulae, then $\neg\phi_1, \phi_1 \vee \phi_2$, and $\phi_1 \wedge \phi_2$ are Boolean formulae.

The truth of $q_i \in Q$ is determined by the run r_σ . For $C \subseteq Q$ define the assignment $q_i = \text{true}$ provided $q_i \in C$. Let $\phi[C]$ denote the truth value of ϕ under this assignment. We say C is *accepting* or *positive* if $\phi[C] = \text{true}$, and is *rejecting* or *negative* otherwise. The *polarity* of a SCS S is “+” if S is accepting, and “-” if S is rejecting.

Definition 2.2.2 *The language generated or accepted or realized by the ω -automaton $\mathcal{A} = \langle T, \phi \rangle$, denoted $L(\mathcal{A})$, is $\{\sigma \mid \sigma \in \Sigma^\omega \text{ such that for some run } r_\sigma \phi[\text{inf}(r_\sigma)] = \text{true}\}$.*

Note, if $\mathcal{A} = \langle T, \phi \rangle$ is a DOA, the DOA accepting $\Sigma^\omega \setminus L(\mathcal{A})$ is $\overline{\mathcal{A}} = \langle T, \neg\phi \rangle$.

Definition 2.2.3 *An ω -language $L \subseteq \Sigma^\omega$ is called ω -regular if $L = L(\mathcal{A})$ for some ω -automaton \mathcal{A} . The class of ω -regular languages is $\{L \mid L \text{ is } \omega\text{-regular}\}$.*

Definition 2.2.4 *We define various types of Boolean formulae which are used to define acceptance conditions [16, 45]:*

1. A **disjunctive formula (DF)**, also called a **Buchi formula**, is a disjunction of Boolean variables, i.e., $\phi = q_1 \vee \dots \vee q_k$; if $S = \{q_1, q_2, \dots, q_k\}$, we write S to denote the disjunctive formula $q_1 \vee \dots \vee q_k$ corresponding to the subset $S \subseteq Q$.
2. A **Co-Buchi formula**, is $\neg\overline{F}$, where \overline{F} is the DF for the subset $Q \setminus F$.
3. A **Muller formula** is $\bigvee_{F \in \mathcal{F}} (\bigwedge_{f \in F} (f) \wedge_{q \notin F} (\neg q))$, where $\mathcal{F} \subseteq 2^Q$.
4. A **Rabin formula** is $\bigvee_{i=1}^n (L_i \wedge \neg(\overline{U}_i))$, where $L_i, U_i, 1 \leq i \leq n$ are DF.
5. A **Streett formula** is $\bigwedge_{i=1}^n (L_i \vee \neg(\overline{U}_i))$, where $L_i, U_i, 1 \leq i \leq n$ are DF.
6. A **Rabin Chain formula** is $\bigvee_{i=1}^n (\neg E_i \wedge F_i)$ where $F_i, E_i, 1 \leq i \leq n$ are DF, and $(\bigwedge_{i=1}^n (E_i \Rightarrow F_i)) \wedge (\bigwedge_{i=1}^{n-1} (F_i \Rightarrow E_{i+1}))$.
7. A **Streett Chain formula** is $\bigwedge_{i=1}^n (F_i \vee \neg E_i)$, where $F_i, E_i, 1 \leq i \leq n$, are DF, and $(\bigwedge_{i=1}^n (F_i \Rightarrow E_i)) \wedge (\bigwedge_{i=1}^{n-1} (E_i \Rightarrow F_{i+1}))$.

Each of the above defined Boolean formulae defines an acceptance condition and an ω -automaton by the same name.

A disjunctive or Buchi formula identifies a subset of states. Therefore, the final states of a $*$ -automaton may be viewed as defining a Buchi formula, and so the states identified by the Buchi formula have come to be called final states (a more appropriate term would perhaps be recurrent states). In a Buchi automaton, a SCS

would be accepting provided it contains a final state; accepting runs visit final states infinitely often.

Suppose $\mathcal{A} = \langle T, F \rangle$ is a deterministic Buchi automaton (DBA). Let $L(\mathcal{A}) = L_\omega$, and viewed as a DFA let $L_*(\mathcal{A}) = L_*$. Since T is deterministic, each ω -word σ has a unique run on T . If $\sigma \in L_\omega$, the unique run r_σ visits a final state infinitely often, and hence infinitely many prefixes of σ are in L_* . Therefore, ω -word σ is in L_ω iff infinitely many prefixes of σ are in L_* . This is written as $L_\omega = \lim L_*$. For example, the automaton of Figure 2.1 viewed as a DBA has the language consisting of all ω - $\{a, b\}$ -words with infinite occurrences of a ; let Y denote this language.

So the language of a DBA is of the form $\lim W$, for some $*$ -regular language W . However, the same is not true for all ω -regular languages. Let us consider the complement of the language of the DBA shown in Figure 2.1, $Z = \bar{Y} = \Sigma^\omega \setminus Y$; this language consists of all ω -words with finite occurrences of a . It is well known [45, 29, 52] that this language is not realizable by a DBA. Assume otherwise for contradiction. Since $b^\omega \in Z$, the run will visit some final state (say f_1) infinitely often, and therefore for some index i_1 the run is at f_1 . Now, consider $b^{i_1}ab^\omega \in Z$. Its run has to visit some final state infinitely often. Therefore for some index $i > i_1$ the run is at a final state (say) f_2 for word $b^{i_1}ab^{i_2}$. Now $f_2 \neq f_1$, since otherwise $b^{i_1}(ab^{i_2})^\omega$ would be accepted. Proceeding in this manner we deduce by a pumping argument that any DBA accepting Z has to have an infinite number of states.

However, Z is accepted by a deterministic Co-Buchi automaton (DCBA). Since a Co-Buchi formula, $\neg \bar{F}$, is the negation of a disjunctive formula, i.e., the complement of the Buchi condition, a set S satisfies the Co-Buchi acceptance condition provided $S \subseteq F$. The Co-Buchi acceptance condition identifies a subset of the states called the *co-final* or persistence set, and any SCS is accepted provided it is contained in the persistence set; a run is accepting provided it finally never leaves the co-final set. In particular, the language Z is accepted by the Co-Buchi automaton with the transition structure of Figure 2.1 and co-final set $\{B\}$.

Thus neither DBA nor DCBA are closed under complement; and, they accept a strict subset of the class of ω -regular languages. It is a simple exercise to construct a nondeterministic Buchi automaton (NBA) to accept Z , and hence not all NBA can

be determinized into DBA. Every ω -regular language is accepted by some NBA.

In a Muller formula, $\bigvee_{F \in \mathcal{F}} (\bigwedge_{f \in F} (f) \wedge_{q \notin F} (\neg q))$ each accepted set is explicitly identified as a minterm; thus the Muller formula is the minterm canonical form for the acceptance condition, and each set identified in the formula is called an *acceptance set*. Any acceptance condition is readily translated into the Muller condition, by enumerating the positive SCSs in the automaton. For instance, the language Y is realized by the transition structure of Figure 2.1 with acceptance sets $\{A, B\}$ and $\{A\}$. McNaughton showed that every ω -regular language can be accepted by a deterministic Muller automaton (DMA) [52].

In a Rabin formula $\bigvee_{i=1}^n (L_i \wedge \neg(\overline{U}_i))$, each disjunct $L_i \wedge \neg\overline{U}_i$, is the conjunction of a Buchi, L_i , and a Co-Buchi formula, $\neg\overline{U}_i$, and identifies a pair of subsets of the states, (L_i, U_i) . A set $S \subseteq Q$ satisfies the Rabin formula, provided it *satisfies some* pair, (L_i, U_i) . S satisfies pair (L_i, U_i) , provided it “touches” L_i , i.e., $S \cap L_i \neq \emptyset$, and is “trapped” in U_i , i.e., $S \subseteq U_i$. If S does not satisfy a pair, it is said to *violate* the pair. Rabin formulae clearly subsume Buchi and Co-Buchi formulae.

In a Streett formula, $\bigwedge_{i=1}^n (L_i \vee \neg(\overline{U}_i))$, each conjunct, $L_i \vee \neg\overline{U}_i$, is the disjunction of a Buchi and a Co-Buchi formula, and identifies a pair (L_i, U_i) . A set $S \subseteq Q$ satisfies the Streett acceptance condition, provided for *each* pair (L_i, U_i) , S either “touches” L_i or is “trapped” in U_i . Streett formulae also clearly subsume Buchi and Co-Buchi formulae.

The negation of a Rabin formula is a Streett formula, and vice-versa. For instance, $\neg \bigvee_{i=1}^n (L_i \wedge \neg(\overline{U}_i)) = \bigwedge_{i=1}^n (\overline{U}_i \vee \neg L_i)$; Rabin pair (L_i, U_i) gets transformed to Streett pair $(\overline{U}_i, \overline{L}_i)$. The Rabin and Streett acceptance conditions are thus syntactically complementary; a set S satisfies Rabin formula ϕ iff it does not satisfy Streett formula $\neg\phi$. Therefore, if $\mathcal{A} = \langle T, \phi \rangle$ is a deterministic Rabin automaton (DRA), then the deterministic Streett automaton $\mathcal{B} = \langle T, \neg\phi \rangle$ has complementary language, $L(\mathcal{B}) = \overline{L(\mathcal{A})}$.

Remark: Our syntax for the Rabin and Streett condition differs from the standard definition in the literature, where a Rabin formula is $\bigvee_{i=1}^n (L_i \wedge \neg U_i)$; a run is accepting (for Rabin acceptance), if for some i , it visits L_i (the GREEN states) infinitely often,

but U_i (the RED states) only finitely often. Complementing U_i translates between the standard syntax and ours. Also, in the standard syntax, the Streett condition $\bigwedge_{i=1}^n (L_i \vee \neg U_i)$ is often written as $\bigwedge_{i=1}^n (U_i \Rightarrow L_i)$.

The Rabin chain formula, $\bigvee_{i=1}^n (\neg E_i \wedge F_i)$, is an instance of a Rabin formula in the traditional syntax, where, in addition the sets E_i, F_i form a chain under set inclusion, i.e., $E_1 \subset F_1 \subset E_2 \subset F_2 \subset \dots \subset E_n \subset F_n$. If a set S satisfies a Rabin chain formula, it satisfies exactly one disjunct (pair). The negation of a Rabin chain formula is also a Rabin chain formula:

$$\neg \bigvee_{i=1}^n (\neg E_i \wedge F_i) = (E_1) \vee \bigvee_{i=1}^{n-1} (\neg F_i \wedge E_{i+1}) \vee (\neg F_n).$$

The Streett chain formula, $\bigwedge_{i=1}^n (F_i \vee \neg E_i)$, is a Streett formula (in the traditional) syntax where, in addition the sets F_i, E_i form a chain under set inclusion, $F_1 \subseteq E_1 \subseteq F_2 \subseteq E_2 \subseteq \dots \subseteq F_n \subseteq E_n$. If set S satisfies the Streett chain formula, then either $S \subseteq \overline{E_n}$, or $S \cap F_1 \neq \emptyset$, or $S \cap F_j \neq \emptyset$ for some $j \in \{2, \dots, n\}$ and $S \subseteq \overline{E_{j-1}}$. The negation of a Streett chain formula may be written as a Streett chain formula.

Also, clearly the negation of a Rabin chain formula is a Streett chain formula and vice-versa. We use the acronym DRCA for deterministic Rabin chain automata, and DSCA for deterministic Streett chain automata. Since we can readily translate a Rabin chain formula to a Streett chain formula, and the Rabin and Streett chain formulae are syntactically complementary, we use DCA for deterministic chain automata when we don't care particularly whether it is Rabin or Streett chain.

Although not every ω -regular can be realized by a DBA or a Co-DBA, for every ω -regular language there exists some NBA, DMA, DRA, DSA, DRCA, or DRCA that accepts it. The class of ω -regular languages is closed under union, intersection, complement, and projection [52]. For all acceptance conditions we have discussed, deciding if the language is empty is in polynomial time [16], while allowing other general Boolean formulae leads to the emptiness problem becoming NP-hard.

Given two ω -automata \mathcal{A} and \mathcal{B} , we say they are *equivalent* provided $L(\mathcal{A}) = L(\mathcal{B})$. Given an ω -automaton $\mathcal{A} = \langle T, \phi \rangle$ we are interested in *translating* it into an equivalent ω -automaton $\mathcal{B} = \langle T', \phi' \rangle$ with a different acceptance condition. As we observed

earlier, the Buchi condition is readily translatable to any other acceptance condition (except Co-Buchi). Also, any acceptance condition is readily translated to the Muller condition, and the Rabin and Streett chain conditions are easily translated between.

Translating between other pairs of acceptance conditions may require changing the transition structure. The issue here is the size of the equivalent automaton \mathcal{B} as a function of the size of the automaton \mathcal{A} . The size of a Buchi automaton is measured by just the size of the transition structure, while for the other acceptance conditions, the number of pairs or acceptance sets is also critical. The size of the equivalent automata, especially for deterministic automata, is determined by two important characteristics: *alternation* and *closure properties*, which we describe next.

For an ω -automaton $\mathcal{A} = \langle T, \phi \rangle$, define \mathcal{C} to be the set of Strongly Connected Subsets (SCSs) of T . There is a partial order on \mathcal{C} induced by set inclusion (\subset).

A *chain* is a linearly ordered set $S_1 \subset S_2 \subset \dots \subset S_m$, where each S_i is a SCS. In an *alternating chain*, the polarity of S_{i+1} is opposite of S_i . The *index* of an alternating chain of length m is $\lceil \frac{m}{2} \rceil$. A positive (negative) chain is an alternating chain in which the polarity of the first set in the chain, S_1 , is “+” (“-”). The *Rabin Index* (RI) is the index of the longest positive chain. The *Streett Index* (SI) is the index of the longest negative chain. The difference between the SI and RI (and vice-versa) is clearly at most one.

Figure 2.2 shows the SCSs of an ω -automaton. An edge from i to j indicates that $i \subset j$. The longest negative chain is (1, 2, 5, 6), and the longest positive chain is (4, 5, 6). Therefore, the RI and the SI are both 2.

Let $K \subseteq \mathcal{C}$. We say K is *superset closed* if for any SCS $S_1 \in K$, every superset, i.e., SCS S_2 such that $S_1 \subset S_2$, is also in K , i.e., $S_2 \in K$. Similarly define $K \subseteq \mathcal{C}$ to be *subset closed*. For $S \in \mathcal{C}$, we say S is *superset (subset) closed* if every superset (subset) of S in \mathcal{C} has the same polarity as S . In Figure 2.2, the positive SCSs 6, 7, and 8 are superset closed.

In a Buchi automaton, the set of positive (negative) SCSs is superset (subset) closed and each positive (negative) SCS is superset (subset) closed; whereas in a Co-Buchi automaton the set of positive (negative) SCSs is subset (superset) closed and each positive (negative) is subset (superset) closed. The RI and SI of Buchi and

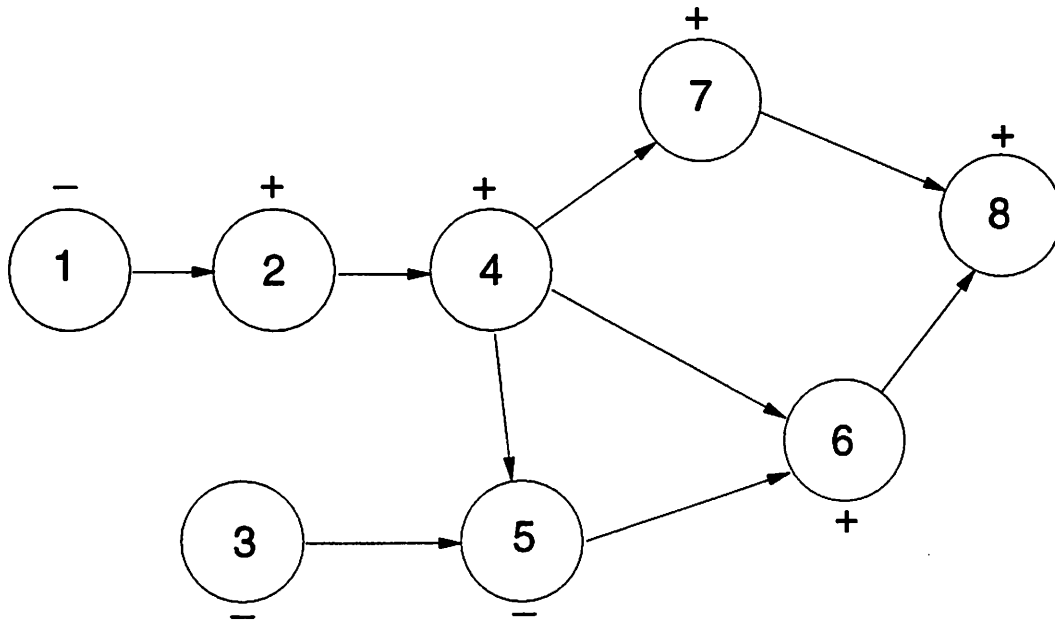


Figure 2.2: Hasse Diagram of SCSs of an automaton

Co-Buchi automata is 1.

Definition 2.2.5 *Given a DOA $\mathcal{A} = \langle T, \phi \rangle$, we say \mathcal{A} is **positive union closed** (**negative union closed**) provided for SCSs $S_1 \subseteq Q$, $S_2 \subseteq Q$, if both S_1 and S_2 are positive (negative) then $S_1 \cup S_2$, if a SCS, is positive (negative).*

From the form of the Boolean formula defining the acceptance conditions, it follows that (just syntactically):

1. The Buchi and Co-Buchi acceptance conditions are both positive and negative union closed.
2. The Streett acceptance condition is positive union closed; the Rabin condition is negative union closed.
3. The chain acceptance condition is both positive and negative union closed.

Although, for instance, a Streett formula may not be negative union closed, the SCSs in the transition structure of a Streett automaton may be such that it is negative union closed.

In general, the Rabin and Streett indices (i.e., the alternation) of deterministic ω -automata, and their closure properties are a function of the form of the acceptance condition, the size of the acceptance condition (number of pairs, acceptance sets), and the transition structure (what subsets are strongly connected)—we use the term **Structural Complexity** of the ω -automaton to collectively refer to these characteristics.

The relative expressiveness of the different acceptance conditions, the structural complexity of ω -automata, and the relationship of $*$ -regular and ω -regular languages have a strong connection to the topological complexity of these sets.

2.3 Topology on Σ^ω

The languages identified by different finite state automata can be identified with levels in the Borel hierarchy in the space of Σ^ω .

Consider the following distance function $d : \Sigma^\omega \times \Sigma^\omega \rightarrow \mathfrak{R}$ on Σ^ω defined as follows:

$$d(\alpha, \beta) = \begin{cases} 0 & \text{if } \alpha = \beta \\ \frac{1}{2^n} & \text{where } n = \min\{i \mid \alpha(i) \neq \beta(i)\} \end{cases}$$

The resulting metric is called the Cantor metric, and the resulting metric topology on Σ^ω is called the Cantor topology. The set Σ^ω endowed with the Cantor topology is called the Cantor space [52, 54]. The basis of the topology is thus all $\frac{1}{2^n}$ -neighborhoods of all sequences $\alpha \in \Sigma^\omega$. For a given ω -word $\alpha \in \Sigma^\omega$, its $\frac{1}{2^n}$ -neighborhood consists of all ω -words that share an n -length prefix with α , i.e., the set $\alpha[0 \dots (n-1)]\Sigma^\omega$. An open set would then be a set of the form $W\Sigma^\omega$, where $W \subseteq \Sigma^*$: $\beta \in W\Sigma^\omega$ if for some $k \in \omega$, $\beta[0]\beta[1] \dots \beta[k] \in W$. It follows then that for every $V \subseteq \Sigma^*$ there exists a closed set consisting of all ω - Σ -words each of which has all its prefixes in V .

In classical terminology the class of open ω -languages is denoted by G and the closed languages by F . The Borel hierarchy is obtained by alternately taking countable intersections and unions starting from the languages in G and F respectively. G_δ (F_σ) denotes the class consisting of countable intersections (unions) of sets in G (F), and similarly $G_{\delta\sigma}$ denotes the class consisting of countable unions of sets in G_δ ,

and so on.

The inclusions depicted in Figure 2.3, where $A \rightarrow B \Leftrightarrow A \subseteq B$, and $A \vec{\rightarrow} B \Leftrightarrow L \in A \text{ iff } \bar{L} \in B$, can be easily derived (see [54] for details).

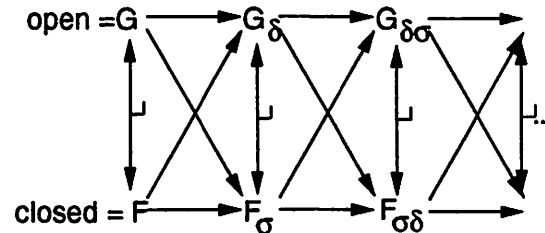


Figure 2.3: Borel hierarchy in Cantor space

The class of open and closed languages are based on finite properties of the words. For every open ω -language L_ω , there exists a $*$ -language L_* such that an ω -word is in L_ω if and only if *some* prefix of it is in L_* . Similarly, for every closed ω -language L_ω , there exists a $*$ -language L_* such that an ω -word is in L_ω if and only if *every* prefix of it is in L_* .

It can be shown that an ω -language L_ω is in G_δ if and only if $L_\omega = \lim W$ for some $*$ -language W . Therefore an ω -regular language L is in G_δ (F_σ) iff L is accepted by a DBA (Co-DBA). Also, it can be shown that every ω -regular language is in the class $G_{\delta\sigma} \cap F_{\sigma\delta}$ [52, 54]. Thus the regular ω -languages occur very low in the Borel hierarchy on the Cantor space.

Also, as we see in the subsequent sections, when we specialize to ω -regular languages the inclusions indicated in Figure 2.3 are easy to see. In the sequel we shall not be concerned with non-regular ω -languages. In the subsequent sections we consider the translation of DOA of increasing complexity.

2.4 Open and Closed ω -regular languages

The Buchi and Co-Buchi acceptance conditions are the simplest acceptance conditions, and we show that any open or closed language presented as a nondeterministic ω -automaton can be realized equivalently either as a Buchi or a Co-Buchi automaton

on “essentially” the same transition structure.

Let L be an open language presented as a nondeterministic Muller automaton $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$. Recall, an open language is of the form $V\Sigma^\omega$, where $V \subseteq \Sigma^*$. Let $F = \{q \mid L(\langle T' = (Q, q, \Sigma, \delta), \phi \rangle) = \Sigma^\omega\}$, i.e., these are the states in \mathcal{A} from which all ω -words are accepted. Delete from δ any triple (f, σ, q) where $f \in F$ and $f \neq q$, and add to δ triples (f, σ, f) , for each $f \in F$ and $\sigma \in \Sigma$, i.e., add a self-loop under every letter of the alphabet for every state in F . Let the new transition relation be denoted by δ' ; in δ' the only edges for $f \in F$ are self loops.

Proposition 2.4.1 *Suppose $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$ is a NMA for an open language. Then the automaton $\mathcal{A}' = \langle T' = (Q, q_0, \Sigma, \delta'), \phi' \rangle$, where ϕ' either denotes the Buchi final set F , or the Co-Buchi final set F , i.e., $\phi = F$ or $\phi = \neg\overline{F}$, and δ' and F are as defined above, accepts the same language as \mathcal{A} .*

Similarly consider a NMA $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$ representing a closed language. Recall that corresponding to a closed ω -language L_ω is a $*$ -language L_* such that $\sigma \in L_\omega$ iff every prefix of σ is in L_* , and if some prefix of σ is not in L_* then $\sigma \notin L_\omega$. If E is the set of states from which the language generated is empty, i.e., $E = \{q \mid L(\langle T' = (Q, q, \Sigma, \delta), \phi \rangle) = \emptyset\}$, delete edges out of states in E and add a self-loop for each state in E under each letter of the alphabet. Suppose δ' is the new transition relation, consider $\mathcal{A}' = \langle T' = (Q, q_0, \Sigma, \delta'), \phi' \rangle$ with ϕ' denoting either Buchi final set $Q \setminus E$, or Co-Buchi Co-final set $Q \setminus E$; then, $L(\mathcal{A}) = L(\mathcal{A}')$ (see also [3]).

Thus every closed or open regular ω -language, represented by a nondeterministic ω -automaton, can be equivalently realized with any acceptance condition on essentially the same transition structure. Since language emptiness for ω -automata is decidable in polynomial time, the equivalent NBA for a closed ω -regular language can be computed efficiently. However, since language universality, i.e., is $L(\mathcal{A}) = \Sigma^\omega$, is PSPACE-hard, the equivalent NBA for open ω -regular languages cannot be computed efficiently.

From this section on we will mostly discuss deterministic ω -automata, for reasons that become apparent as the chapter progresses (there isn't a hierarchy on the ω -

regular languages induced by restricting the number of pairs in nondeterministic Rabin/Streett automata).

2.5 G_δ languages–DBA-realizable languages

As we had seen earlier the language $(a + b)^*b^\omega$ cannot be realized as a DBA. Landweber has given a structural characterization of when a DOA can be equivalently realized as a DBA.

Theorem 2.5.1 (Landweber [29]) *Let $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$ be a DMA.*

There is an equivalent DBA to \mathcal{A} if and only if every accepting SCS is superset closed.

Proof: “ \Rightarrow ”: Let S be a positive SCS, and T be a negative SCS containing S . It can be shown by a pumping argument, similar to the proof that the language $(a + b)^*b^\omega$ cannot be realized as a DBA, that the language of the DOA cannot be realized as a DBA.

“ \Leftarrow ”: Let \mathcal{A} be a DMA such that the set of acceptance sets \mathcal{F} is superset closed. The equivalent DBA $\mathcal{A}' = \langle T', F' \rangle$, where $T' = (Q \times 2^Q, (q_0, \{q_0\}), \Sigma, \delta')$; $\delta'((q, M), \sigma) = (q', M')$ where $q' = \delta(q, \sigma)$ and

$$M' = \begin{cases} \emptyset & \text{if } q' \cup M \supseteq S \in \mathcal{F} \\ q' \cup M & \text{otherwise.} \end{cases}$$

The final state-set, F' , of the DBA \mathcal{A}' is $Q \times \emptyset$.

M remembers the states visited since the last time an acceptance set was included in the run. M will be reset infinitely often, i.e., F' visited infinitely often, iff an acceptance set is included in the run infinitely often. ■

Definition 2.5.1 *Given a DOA $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$, define the **superset closed sublanguage** of \mathcal{A} , $\text{supLan}(\mathcal{A})$, to be $\{\sigma \mid \sigma \in L(\mathcal{A}) \text{ and } \text{inf}(r_\sigma) \text{ is superset closed}\}$.*

Thus the superset closed sublanguage is the set of sequences in the language with superset closed infinity sets.

Definition 2.5.2 A DOA \mathcal{A} is **DBA-realizable** if there exists an equivalent DBA \mathcal{A}' . From the topological characterization, we can also say \mathcal{A} is in G_δ or $L(\mathcal{A}) \in G_\delta$ if \mathcal{A} is DBA-realizable; this (by Landweber's Thm.) is also equivalent to saying that $\text{supLan}(\mathcal{A}) = L(\mathcal{A})$.

The question arises: if given two equivalent DOA \mathcal{A}_1 and \mathcal{A}_2 , is $\text{supLan}(\mathcal{A}_1) = \text{supLan}(\mathcal{A}_2)$?

Proposition 2.5.2 For a DOA \mathcal{A} , $\text{supLan}(\mathcal{A})$ is not an invariant of $L(\mathcal{A})$; i.e., there exists equivalent DOA \mathcal{A}_1 and \mathcal{A}_2 such that $\text{supLan}(\mathcal{A}_1) \neq \text{supLan}(\mathcal{A}_2)$.

Proof: DRA \mathcal{A}_1 is defined by the transition structure on the left in Figure 2.4, and

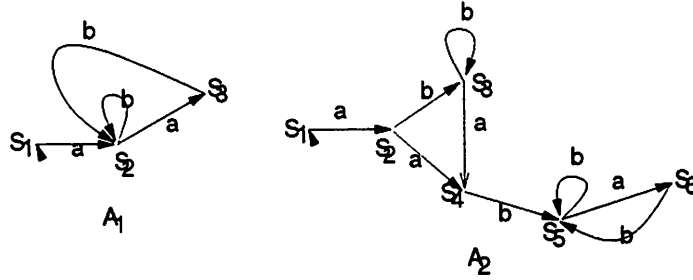


Figure 2.4: supLan not an invariant of the language

acceptance condition comprised of a single Rabin-pair $(\{S_2\}, \{S_2\})$.

DRA \mathcal{A}_2 is defined by the transition structure on the right in Figure 2.4, and acceptance condition comprised of Rabin pairs $(\{S_3\}, \{S_3\})$ and $(\{S_5\}, \{S_5\})$.

$L(\mathcal{A}_1) = L(\mathcal{A}_2)$, but $ab^\omega \notin \text{supLan}(\mathcal{A}_1)$ because (in \mathcal{A}_1) $\{S_2\}$ is not superset closed, whereas $ab^\omega \in \text{supLan}(\mathcal{A}_2)$ because (in \mathcal{A}_2) $\{S_3\}$ is superset closed. ■

We now consider the question of constructing the equivalent DBA for DOA with different acceptance conditions. In the proof of Theorem 2.5.1, to construct the equivalent DBA we had to augment the state space of the automaton by an auxiliary set $M \in 2^Q$ that remembered the states visited since the last time an acceptance set was completed. We call this auxiliary set memory. We shall be interested in minimizing the size of this memory. In certain cases, we can do without any memory, by identifying final states on the transition structure.

Definition 2.5.3 Given an ω -automaton $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$, a state $q \in Q$ is termed **final** provided every SCS C containing q is accepting.

The terminology, final state, comes from Buchi automata where every SCS containing a final state is accepting; every accepting SCS is superset closed.

Definition 2.5.4 An ω -automaton, $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$, is **Buchi-type (BT)** if there exists a subset $F \subseteq Q$ such that the Buchi automaton, $\mathcal{B} = \langle T, F \rangle$ is equivalent to \mathcal{A} .

Theorem 2.5.3 Let $\mathcal{A} = \langle T, \phi \rangle$ be a DOA that is negative union closed. Suppose $\mathcal{A} \in G_\delta$, i.e., $L(\mathcal{A})$ is DBA-realizable. Then, $L(\mathcal{A})$ is equal to the language of the DBA $\mathcal{A}' = \langle T, F \rangle$, where F is the set of final states of \mathcal{A} .

Proof: Assume that there is a positive SCS S in \mathcal{A} such that $S \cap F = \emptyset$.

By assumption $q \in S \Rightarrow q \notin F$. Thus for every $q \in S$, there exists a SCS C_q such that C_q is negative. Now consider $C = \cup_{q \in S} C_q$. Since $S \subseteq C$, and each C_q is a SCS, C is a SCS. C is negative since it is a union of negative SCSs. This contradicts \mathcal{A} being in G_δ , because S , a positive SCS, should be superset closed. ■

Since the Rabin, Chain, and Co-Buchi conditions are negative union closed we have:

Theorem 2.5.4 Let $\mathcal{A} = \langle T, \phi \rangle$ be either a DRA, DCA, or DCBA. Then \mathcal{A} is in G_δ if and only if \mathcal{A} is Buchi-type. Also, $\text{supLan}(\mathcal{A}) = L(\mathcal{A}')$ where, \mathcal{A}' is the DBA on transition structure T and final state set the final states of \mathcal{A} .

However, neither the Muller nor the Streett condition is negative union closed and:

Proposition 2.5.5 There exists DMA & DSA which are DBA-realizable but not Buchi-type.

Proof: The example of Fig. 2.5 is a DMA that is not Buchi-type (because no state is final) but whose language is generated by a DBA. The same automaton is a counter-example for DSA as well, with Streett pairs $\{(\{1\}, \emptyset), (\{2\}, \emptyset)\}$. ■

But, for a one pair DSA the superset closed sublanguage can be realized as a DBA on the same transition structure.

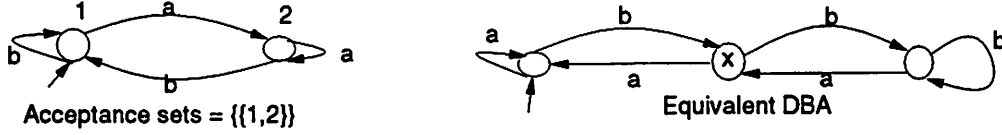


Figure 2.5: DMA not Buchi-type but DBA-realizable

Lemma 2.5.6 For DSA $\mathcal{A}_i = \langle T = (Q, q_0, \Sigma, \delta), L_i \vee \neg(\overline{U}_i) \rangle$, an accepting SCS C is superset closed if and only if $C \cap F_i \neq \emptyset$, where F_i is the set of final states of \mathcal{A}_i .

Proof: Note that $q \in F_i$ provided either $q \in L_i$ or after deleting L_i , $SCC(q) \subseteq U_i$.

Consider a superset closed accepting SCS C . Assume for contradiction that $C \cap F_i = \emptyset$; therefore $C \cap L_i = \emptyset$. Consider the subgraph of STG T on vertex set $Q \setminus F_i$. Since C is an accepting SCS, $C \subseteq U_i$. Since C is superset closed, $SCC(C) \subseteq U_i$, and therefore every state $s \in C$ is final, i.e., $s \in C \Rightarrow s \in F_i$. ■

Theorem 2.5.7 Given DSA $\mathcal{A} = \langle T, \bigwedge_{i=1}^h (L_i \vee \neg(\overline{U}_i)) \rangle$, an accepting SCS C is superset closed if and only if for each i , $1 \leq i \leq h$, $C \cap F_i \neq \emptyset$, where F_i is the set of final states of $\mathcal{A}_i = \langle T, L_i \vee \neg(\overline{U}_i) \rangle$.

Proof: If for each i , $C \cap F_i \neq \emptyset$, then C is superset closed in \mathcal{A} since C is superset closed for each pair, i.e., in each \mathcal{A}_i .

Conversely, assume C is superset closed. Since $L(\mathcal{A}) = \bigcap_i L(\mathcal{A}_i)$, considering each pair (L_i, U_i) , C is superset closed. By Lemma 2.5.6, $C \cap F_i \neq \emptyset$ for each i . ■

Given a DSA $\mathcal{A} = \langle T, \bigwedge_{i=1}^h (L_i \vee \neg(\overline{U}_i)) \rangle$ we apply Theorem 2.5.7 to construct a DBA with language $supLan(\mathcal{A})$, the DBA for $\langle T, \bigwedge_{i=1}^h F_i \rangle$. The basic idea is to make h copies of the transition structure and direct the transitions from the states in F_i in the i^{th} copy to the corresponding states in the $(i+1)^{st}$ copy.

DSA2DBA Construction

INPUT: DSA $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \bigwedge_{i=1}^h (L_i \vee \neg(\overline{U}_i)) \rangle$

OUTPUT: DBA $\mathcal{A}' = \langle T' = (Q', q'_0, \Sigma, \delta'), S \rangle$, where

F_i is the set of final states of $\langle T, L_i \vee \neg(\overline{U}_i) \rangle$

$Q' = Q \times \{1, \dots, h\}$, $q'_0 = (q_0, 1)$

$$\delta'((q, j), a) = \begin{cases} (\delta(q, a), j) & \text{if } q \notin F_j \\ (\delta(q, a), j+1) & \text{if } q \in F_j \text{ and } j < h \\ (\delta(q, a), 1) & \text{if } q \in F_h \end{cases}$$

$$S = \cup_{i \in \{1, 2, \dots, h\}} F_i \times i$$

Thus the equivalent DBA for a DSA in G_δ on n states and h pairs has nh states, polynomial in the size of the DSA. This is in contrast to a result of [44] that there is a family of DSA such that any equivalent NBA is of size exponential in the size of the DSA; in G_δ DSA and DBA are polynomially related.

Next we revisit DMA in G_δ . We present another translation of a DMA in G_δ into an equivalent DBA whose size is a function of the number of minimal acceptance sets in the DMA. Let $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$ be a DMA with h minimal positive superset closed SCSs, F_1, F_2, \dots, F_h . Assume that the states of \mathcal{A} are numbered $1, 2, \dots, n$. The equivalent DBA $\mathcal{A}' = \langle T' = (Q', q'_0, \Sigma, \delta'), F' \rangle$, where $Q' = Q \times (Q \cup 0)^h$, $q'_0 = (q_0, 0, 0, \dots, 0)$. $\delta'((q, q_1, q_2, \dots, q_h), \sigma) = (\delta(q, \sigma), q'_1, q'_2, \dots, q'_h)$, where q'_i is defined as follows. Let $q' = \delta(q, \sigma)$.

- If $q_i = 0$ then $q'_i =$ the least numbered state in F_i .
- If $q' = q_i$ then:
 - if q_i is the highest numbered state in F_i , $q'_i = \emptyset$,
 - else q'_i is set to the next numbered state in F_i after q_i .
- If $q_i \neq 0$ and $q' \neq q_i$, then $q'_i = q_i$.

The final state set, is $F' = \{(q, q_1, q_2, \dots, q_h) | q_i = 0 \text{ for some } i\}$. The state q_i records the target state to be reached in the positive SCS F_i , and is advanced every time the target state is attained. If it is set to 0 infinitely often, the set F_i will be included in infinity set of the run.

Thus the equivalent DBA for a DMA in G_δ has $n(n+1)^h$ states where n is the number of states in the DMA, and h is the number of minimal superset closed SCSs (see also [30]). This construction, which we shall refer to as the *structural approach*,

may produce a smaller equivalent DBA than that in the proof of Landweber's theorem, especially when the number of minimal positive superset closed SCSs, h , is small.

The question arises if for DMA and DSA in G_δ there is always an equivalent DBA with a polynomial number of states as the DMA or DSA, and if the constructions defined in this section are optimum in any sense.

2.6 Lower bounds for translating into DBA

We collect a couple of lemmas and definitions first.

Lemma 2.6.1 *Suppose $\sigma \in \Sigma^\omega$ is a periodic sequence, i.e., $\sigma = c^\omega$ for some $c \in \Sigma^+$, $|c| \geq 1$, and $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$ is a DOA. The (unique) run, r_σ , of σ on \mathcal{A} is ultimately periodic with period some multiple of $|c|$, i.e., $r_\sigma = r_\sigma[0] \dots r_\sigma[i|c| - 1](r_\sigma[i|c|] \dots r_\sigma[j|c| - 1])^\omega$, or $r_\sigma = (r_\sigma[0] \dots r_\sigma[j|c| - 1])^\omega$ where $0 \leq i < j < |Q|$, and the run of $c^{(j-i)}$ on \mathcal{A} from $r_\sigma[i|c|]$ is exactly $r_\sigma[i|c|] \dots r_\sigma[j|c|]$.*

Proof: Let $\delta(q_0, c) = q_1$. If $q_1 = q_0$, then r_σ is already periodic and $j = 1$ and $i = 0$, i.e., $r_\sigma = (r_\sigma[0] \dots r_\sigma[|c| - 1])^\omega$. Otherwise, consider $\delta(q_1, c) = q_2$. If $q_2 \neq q_0$ and $q_2 \neq q_1$, then consider $\delta(q_2, c) = q_3$, and so on until $q_j = q_i$ for $i < j$. Since Q is finite we are guaranteed the existence of j and i such that $0 \leq i < j < |Q|$. ■

Definition 2.6.1 *Given a DOA $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$, and a SCS $C \subseteq Q$, and a state $q \in C$ we say a word $c \in \Sigma^+$, describes C starting from q provided $\delta(q, c) = q$ and $\overline{\delta(q, c)} = C$, where $\overline{\delta(q, c)} \stackrel{\text{def}}{=} \{q' | q' = \delta(q, c[0] \dots c[i]) \text{ for some } i, 0 \leq i \leq |c| - 1\}$.*

Definition 2.6.2 *The transition structure T^n is defined on the state set*

$Q_n = \{1, 2, \dots, n\}$, *alphabet $\Sigma_n = \{1, 2, \dots, n\}$, initial state 1, and transition function δ_n , where for $i, j, 1 \leq i, j \leq n, \delta_n(i, j) = j$.*

Lemma 2.6.2 *Let $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$ be a DOA. Let $c \in \Sigma^*$ describe SCS C_1 from q and $d \in \Sigma^*$ describe SCS C_2 from p , and $s \in C_1 \cap C_2 \neq \emptyset$. Then, there exists $f \in \Sigma^*$ such that f describes $C_1 \cup C_2$ from s , and $\text{inf}(f^\omega) = \text{inf}(c^\omega) \cup \text{inf}(d^\omega)$.*

Proof: Let i , $0 \leq i < |c|$, be such that $\delta(q, c[0 \dots i]) = s$. Then $c' = c[i + 1 \dots |c| - 1]c[0 \dots i]$ describes C_1 from s .

Similarly there exists d' describing C_2 from s . $f = c'd'$ will describe $C_1 \cup C_2$ from s , and $\text{inf}(f^\omega) = \text{inf}(c^\omega) \cup \text{inf}(d^\omega)$. ■

Theorem 2.6.3 *For every $n \geq 2$, there exists a DMA on n states such that any equivalent DBA has at least $2^{\lfloor \frac{n}{2} \rfloor}$ states.*

Proof: For every $n \geq 2$, consider the language L_n on $\Sigma_n = \{1, \dots, n\}$, of all ω -words over Σ_n , where at least $\lfloor \frac{n}{2} \rfloor + 1$ distinct numbers appear infinitely often, $L_n = \{\sigma \mid \sigma \in \Sigma_n^\omega \text{ and } |\text{inf}(\sigma)| \geq \lfloor \frac{n}{2} \rfloor + 1\}$.

This language is accepted by the DMA $\mathcal{A}_n = \langle T^n, \phi_n \rangle$, where ϕ_n denotes as acceptance sets any subset S of Q_n such that $|S| \geq (\lfloor \frac{n}{2} \rfloor + 1)$ (see Definition 2.6.2 for T^n).

Let S be a subset of Q_n of size $\lfloor \frac{n}{2} \rfloor$. For $s \in S$, a word $c \in (\Sigma_n)^*$ describes S from s in \mathcal{A}_n , i.e., T_n , if and only if $\text{inf}(c^\omega) = S$.

Let \mathcal{A}'_n be an equivalent DBA. Let the inf of the run of c^ω on \mathcal{A}'_n be S' . By Lemma 2.6.1, $\exists m > 0$, such that c^m describes S' from some $s' \in S'$.

Let S_1 and S_2 be two distinct subsets of Q_n of size $\lfloor \frac{n}{2} \rfloor$. Therefore, there are $c_1, c_2 \in (\Sigma_n)^*$ that describe S_1 and S_2 in T_n respectively from some $s_1 \in S_1$ and $s_2 \in S_2$, such that $\text{inf}(c_1^\omega) = S_1$ and $\text{inf}(c_2^\omega) = S_2$.

Now for c_1^ω and c_2^ω , let the inf sets of their runs in the equivalent DBA \mathcal{A}'_n be S'_1 and S'_2 . The claim is that $S'_1 \cap S'_2 = \emptyset$. Assume otherwise for contradiction. By Lemma 2.6.2, for $s' \in S'_1 \cap S'_2$, there exists $f \in (\Sigma_n)^*$ such that f describes $S'_1 \cup S'_2$ from s' and $\text{inf}(f^\omega) = \text{inf}((c_1^{m_1})^\omega) \cup \text{inf}((c_2^{m_2})^\omega) = \text{inf}(c_1) \cup \text{inf}(c_2)$. Thus the inf of the run of f^ω on \mathcal{A}_n will be $\text{inf}(c_1) \cup \text{inf}(c_2)$, an accepting set since $|\text{inf}(c_1) \cup \text{inf}(c_2)| > \lfloor \frac{n}{2} \rfloor$. But a DBA is negative union closed, and therefore $S'_1 \cup S'_2$ will have negative polarity in \mathcal{A}'_n —a contradiction.

Therefore corresponding to every distinct subset of Q_n of size $\lfloor \frac{n}{2} \rfloor$ there is at least one distinct state in the equivalent DBA \mathcal{A}'_n . Thus any equivalent DBA has at least $\binom{n}{\lfloor \frac{n}{2} \rfloor}$ states, and this is greater than or equal to $2^{\lfloor \frac{n}{2} \rfloor}$. ■

Theorem 2.6.3, implies that the number of states in an equivalent DBA may be exponential in the number of states of a DMA in G_δ . Also, the construction outlined in the proof of Landweber's theorem is essentially (asymptotically) optimal—the lower bound is $\binom{n}{\lfloor \frac{n}{2} \rfloor} \geq 2^{\lfloor \frac{n}{2} \rfloor} = 2^{O(n)}$, and the upper bound is $n2^n = 2^{O(n)}$. The structural approach in this case computes an equivalent DBA with $2^{\log n 2^{\frac{n}{2}}}$ states, doubly-exponential in the number of states of the DMA.

Theorem 2.6.4 *For every n there exists a DSA with n states and $h = \binom{n}{\lfloor \frac{n}{2} \rfloor}$ pairs such that any equivalent DBA has to have at least $\binom{n}{\lfloor \frac{n}{2} \rfloor}$ states.*

Proof: The same language-family as in the proof of Theorem 2.6.3 will suffice. L_n is accepted by DSA $\mathcal{A}_n = \langle T^n, \psi_n \rangle$, where ψ_n denotes the set of Streett pairs $\{(S, \emptyset) \mid S \subseteq \{1, \dots, n\} \text{ and } |S| = \lfloor n/2 \rfloor\}$. ■

Thus, although the equivalent DBA for a DSA in G_δ is of size polynomial in the *size* of the DSA, it could be exponential in the number of states of the DSA. In addition Theorem 2.6.4 establishes that **DSA2DBA** is essentially optimal, the upper and lower bounds differ by a constant factor in the exponent.

2.7 Testing for DBA-realizability

Not every deterministic ω -automaton can be translated into a DBA. In this section we devise algorithms to test if a given DOA is Buchi-type or if it is DBA-realizable but not Buchi-type, and if so to effect the translation into an equivalent DBA discussed in Section 2.5.

2.7.1 Checking for Buchi-typeness

Conceptually, we check if DOA $\mathcal{A} = \langle T, \phi \rangle$ is BT, by finding the DBA $\hat{\mathcal{B}} = \langle T, \hat{F} \rangle$ which has the largest language such that $L(\hat{\mathcal{B}}) \subseteq L(\mathcal{A})$ (note that if for DBA $\mathcal{B} = \langle T, F \rangle$ $L(\mathcal{B}) = L(\mathcal{A})$, then $L(\mathcal{B}) \subseteq L(\hat{\mathcal{B}})$). We then check whether $L(\mathcal{A}) - L(\hat{\mathcal{B}}) = \emptyset$. If so, \mathcal{A} is Buchi-type. Otherwise $L(\hat{\mathcal{B}})$ is a proper subset of $L(\mathcal{A})$ and therefore there is no Buchi automaton on the same transition structure as \mathcal{A} accepting the same

language (thus, \mathcal{A} is not BT), although \mathcal{A} may be DBA-realizable. Algorithmically, checking if a given DOA is BT involves the following two steps:

Step 1: Finding the set of final states, \hat{F} (see Definition 2.5.3). We check if a state s is final by testing for the existence of a negative SCS through s .

Step 2: Checking if $L(\mathcal{A}) - L(\hat{\mathcal{B}}) = \emptyset$. This is not performed by the traditional algorithm for language containment. Since \mathcal{A} and $\hat{\mathcal{B}}$ have the same transition structure, it suffices to delete the states in \hat{F} and then check if the resulting graph contains a positive SCS. If it contains a positive SCS, the DOA \mathcal{A} is not Buchi-type, since the positive SCS discovered did not pass through a final state. Checking for the existence of a positive SCS, is straightforward and algorithms are well known [16].

We describe next how to compute the set of final states F (the non-final states are $N = Q \setminus F$) and check if a given DBA is DBA-realizable (and not Buchi-type).

Muller Automata

Let $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$ be a DMA, where ϕ denotes the set of acceptance sets $\mathcal{F} \subseteq 2^Q$. Define the Haase diagram H on the vertex set \mathcal{F} as follows. There is a directed edge from vertex C_1 to vertex C_2 iff $C_1 \subset C_2$ and there is no acceptance set C_3 such that $C_1 \subset C_3 \subset C_2$. We assign a weight of 1 to the edge $C_1 \rightarrow C_2$ if there is some SCS C_3 sandwiched between C_1 and C_2 , i.e. such that $C_1 \subset C_3 \subset C_2$; if such a SCS C_3 exists, it has negative polarity, and otherwise the edge is assigned a weight of 0.

Determining if edge $E = C_1 \rightarrow C_2$ should be labeled 1 or 0 goes as follows. Let $D = C_2 \setminus C_1$. Edge E is assigned a weight of 1 provided for some $q \in D$, in the subgraph of T on the vertex set $C_2 \setminus q$, the polarity of $SCC(C_1)$ is $-$. Since the SCCs of a directed graph can be computed in linear time, i.e. $O(|T|)$ time, the weight of an edge in the Haase diagram can be determined in $O(|Q||T|)$ time.

We augment the Haase diagram H with a *bottom* vertex corresponding to the empty set ϕ , and a *top* vertex corresponding to the set Q , if Q is not an acceptance set. Add edges from the bottom vertex to the vertices C_m , such that C_m is a minimal acceptance set, and from each maximal acceptance set to the top vertex. If l is

the number of acceptance sets in the DMA \mathcal{A} , then labeling the edges of H takes $O(l^2|Q||T|)$ time.

Define a valuation function on the vertex set of H , H_v , $Val : H_v \rightarrow \omega$ as follows. The top vertex Q is assigned a value of 0. The value of every vertex v , other than Q in H_v is computed as: $Val(v) = \max_{\{u:(v,u) \in E\}} \{w(v,u) + Val(u)\}$, where $w(v,u)$ is the weight of edge (v,u) . Since the value of vertex v is greater than zero iff there exists a negative SCS v' containing v , the set of non-final states, N , is $(\cup_{\{v:Val(v)>0\}} v) \cup (\cup_{\{S:S \text{ is a negative SCC}\}} S)$. The set of final states is $Q \setminus N$.

Since the valuation, Val , can be computed in time linear in the size of the Haase Diagram H , the set of final states can be computed in time $O(l^2|Q||T|)$. DMA \mathcal{A} is in G_δ if the only edges with weight one, if any, are those from the bottom vertex.

Rabin Automata

Without loss of generality let DRA $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \bigvee_{i=1}^h (L_i \wedge \neg \overline{U_i}) \rangle$ have a single SCC C . For the Rabin condition, a SCS is negative if all the pairs are violated. If for each pair (L_i, U_i) , $C \not\subseteq U_i$, then no state in C is final, i.e. $F = \emptyset$ and $N = C$. On the other hand, if $C \subseteq U_i$, then $L_i \subseteq F$. Let $I \subseteq \{1, 2, \dots, h\}$ be the indices of pairs such that $C \subseteq U_j$ for $j \in I$. Then $\cup_{j \in I} L_j \subseteq F$. Consider the SCCs, ψ_1, \dots, ψ_p , of the subgraph of T on the vertex set $C \setminus (\cup_{j \in I} L_j)$. The trivial SCCs amongst the ψ 's are also final states. Recurse on the non-trivial SCCs with pairs indexed by the set $\{1, 2, \dots, h\} \setminus I$.

If no more pairs remain to be considered, the states in the SCC are non-final. The recursive algorithm sketched above to determine the final states of a Rabin automaton has time complexity $O(|T|h^2)$.

If a DRA is not Buchi-type then it is also not in G_δ (Theorem 2.5.4).

Streett Automata

In a Streett automaton, $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \bigwedge_{i=1}^h (L_i \vee \neg \overline{U_i}) \rangle$, a SCS is negative provided it violates at least one of the pairs. For each pair (L_i, U_i) , compute the SCCs of the subgraph of T on the vertex set $Q \setminus L_i$. Any non-trivial SCC not contained in

U_i is comprised of non-final states. Thus the set of states Q can be partitioned into final and non-final states in a Streett automaton in time $O(|T|h)$.

If the given DSA is not Buchi-type is it in G_δ ? We point forward to Section 2.10.1, where we define the *StrExcl* construction. The given DSA \mathcal{A} is in G_δ iff $L(\mathcal{A}') = \emptyset$, and the procedure has time complexity $O(nh^3)$.

2.8 F_σ -languages—Co-DBA realizable languages

Since the classes G_δ and F_σ are complementary, and the Streett and Rabin acceptance are syntactically complementary, the dual of the results of the previous section hold for F_σ -languages.

The language of a DOA is realizable as a Co-DBA iff it is positive subset closed. The language of a positive union closed DOA is in F_σ iff the language can be equivalently realized as a Co-DBA on the same transition structure. The subset closed sublanguage of a positive union closed DOA, and in particular a DSA, can be realized as a DCBA on the same transition structure as the DOA.

There exist efficient algorithms to check if a given DOA is Co-Buchi type, as well as to construct the equivalent DCBA in case the language of a DOA is in F_σ .

2.8.1 $G_\delta \cap F_\sigma$ regular languages

In a DOA that is in $G_\delta \cap F_\sigma$ a positive SCS is both positive superset and subset closed. Therefore a positive SCC can be thought of being comprised entirely of final states or being included in the co-final set, while every SCS in a negative SCC is negative. Thus every DOA in $G_\delta \cap F_\sigma$ can be realized as either a DBA or a DCBA on the same transition structure.

2.9 $G_{\delta\sigma} \cap F_{\sigma\delta}$ — ω -regular languages

In a DBA, the accepting SCSs are superset closed. Not every language can be accepted by a DBA. However, any ω -regular language can be accepted by a deter-

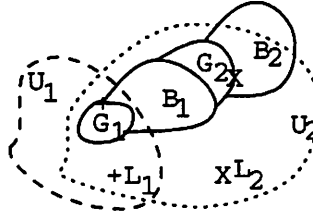


Figure 2.6: Positive chain of index 2 in a 2-pair DRA

ministic Rabin or Streett automaton (DRA or DSA). Even amongst DRA and DSA the number of pairs induces a hierarchy of languages, i.e., there are certain languages that require at least a certain number of pairs to be realized as, say, a DRA.

For instance consider a 2-pair DRA with pairs (L_1, U_1) and (L_2, U_2) . We can have an alternating positive chain $G_1 \subset B_1 \subset G_2 \subset B_2$, as indicated in Figure 2.6 (+ marks states in L_1 and \times marks states in L_2).

It is easy to see that a 2-pair DRA cannot support a positive chain of index greater than 2. It was shown by Wagner [55] and Kaminski [24] that the index of the longest positive (negative) chain is the minimum number of pairs required to realize the automaton equivalently as a DRA (DSA).

We prove this result in a more general form next.

Theorem 2.9.1 (Chain Correspondence) *Let $\mathcal{A}_{q_0} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$ and $\mathcal{A}'_{q'_0} = \langle T' = (Q', q'_0, \Sigma, \delta'), \phi' \rangle$ be two DOA each of whose states is reachable. Given a chain $C_1 \subset C_2 \subset C_3 \subset \dots \subset C_n$ in DOA \mathcal{A}_{q_0} and $q \in C_1$, there exists a chain $C'_1 \subseteq C'_2 \subseteq C'_3 \subseteq \dots \subseteq C'_n$ in $\mathcal{A}'_{q'_0}$, $\alpha, \gamma \in \Sigma^*$, $q' \in C'_1$, and words $c_1, c_2, \dots, c_n \in \Sigma^*$, such that for each pair (C_k, C'_k) , c_k describes C_k from q in \mathcal{A} , and C'_k from q' in $\mathcal{A}'_{q'_0}$, and $\delta(q_0, \alpha\gamma) = q$ and $\delta'(q'_0, \alpha\gamma) = q'$, and $\alpha = \epsilon$ if $q = q_0$, and γ describes C_n in \mathcal{A}_{q_0} from q .*

Proof: The proof is by induction on the length of the chain.

The base case: Since q is reachable in \mathcal{A}_{q_0} apply (say) the shortest string, $\alpha \in \Sigma^*$, from q_0 to reach q , i.e., an α such that $\delta(q_0, \alpha) = q$. Let $\delta'(q'_0, \alpha) = p$. Let $\xi \in \Sigma^*$ describe C_1 from q in \mathcal{A}_{q_0} . Apply $\beta = \xi^\omega$ from p in $\mathcal{A}'_{q'_0}$. By Lemma 2.6.1 the run of β in $\mathcal{A}'_{q'_0}$ will be ultimately periodic, and let the infinity set of the run be C'_1 . For

some positive integer $l > 0$, $\delta'(p, \xi^l) = q' \in C'_1$. Again, by Lemma 2.6.1, for some $0 < m < |Q'|$, $\delta'(q', \xi^m) = q'$ and $\overline{\delta'(q', \xi^m)} = C'_1$, i.e., ξ^m describes C'_1 from q' in $\mathcal{A}'_{q'_0}$. It is clear that ξ^m also describes C_1 from q in \mathcal{A}_{q_0} . Thus the required word is $c_1 = \xi^m$, and $\gamma = \xi^l$.

Induction step: Consider chain $C_1 \subset C_2 \subset C_3 \subset \dots \subset C_n \subset C_{n+1}$, and $q \in C_1$, in \mathcal{A}_{q_0} . We shall repeatedly apply the induction hypothesis to the prefix of the chain: $K_n = C_1 \subset C_2 \subset C_3 \subset \dots \subset C_n$. Let ξ_{n+1} describe C_{n+1} from q in \mathcal{A}_{q_0} .

By the induction hypothesis for K_n , q , and \mathcal{A}_{q_0} , and $\mathcal{A}'_{q'_0}$, there exists $\alpha, \gamma_1 \in \Sigma^*$ and chain $C_1^1 \subseteq C_2^1 \subseteq C_3^1 \subseteq \dots \subseteq C_n^1$, $q_1^1 \in C_1^1$, and $c_1^1, c_2^1, \dots, c_n^1 \in \Sigma^*$ such that:

1. γ_1 describes C_n in \mathcal{A}_{q_0} from q .
2. $\delta(q_0, \alpha\gamma_1) = q$ and $\delta'(q'_0, \alpha\gamma_1) = q_1^1$.
3. For i , $1 \leq i \leq n$, c_i^1 describes C_i from q in \mathcal{A}_{q_0} and C_i^1 from q_1^1 in $\mathcal{A}'_{q'_0}$.

Next apply ξ_{n+1} from q_1^1 . Let $\delta'(q_1^1, \xi_{n+1}) = q_2^1$.

If $q_2^1 \neq q_1^1$, then applying the induction hypothesis for K_n , q , and \mathcal{A}_q , and $\mathcal{A}'_{q'_2}$, there exists $\gamma_2 \in \Sigma^*$ and chain $C_1^2 \subseteq C_2^2 \subseteq C_3^2 \subseteq \dots \subseteq C_n^2$, $q_1^2 \in C_1^2$, and $c_1^2, c_2^2, \dots, c_n^2 \in \Sigma^*$ such that:

1. γ_2 describes C_n in \mathcal{A}_{q_0} from q .
2. $\delta'(q_2^1, \gamma_2) = q_1^2$.
3. For i , $1 \leq i \leq n$, c_i^2 describes C_i from q in \mathcal{A}_{q_0} and C_i^2 from q_1^2 in $\mathcal{A}'_{q'_0}$.

If $q_1^2 \notin \{q_1^1, q_2^1\}$, then let $\delta'(q_1^2, \xi_{n+1}) = q_2^2$.

If $q_2^2 \notin \{q_1^1, q_2^1, q_1^2\}$, we continue the process alternately applying the induction hypothesis for K_n , q , and \mathcal{A}_q , and $\mathcal{A}'_{q'_j}$, getting $\gamma_j \in \Sigma^*$ and chain $C_1^j \subseteq C_2^j \subseteq C_3^j \subseteq \dots \subseteq C_n^j$, $q_1^j \in C_1^j$, and $c_1^j, c_2^j, \dots, c_n^j \in \Sigma^*$ such that:

1. γ_j describes C_n in \mathcal{A}_{q_0} from q .
2. $\delta'(q_2^{j-1}, \gamma_j) = q_1^j$.
3. For i , $1 \leq i \leq n$, c_i^j describes C_i from q in \mathcal{A} and C_i^j from q_1^j in $\mathcal{A}'_{q'_0}$.

If $q_1^j \notin \{q_1^1, q_2^1, q_1^2, q_2^2, q_1^3, q_2^3, \dots, q_1^{j-1}, q_2^{j-1}\}$, then let $\delta'(q_1^j, \xi_{n+1}) = q_2^j$.

And so on. Since Q' is a finite set, we reach an index k such that one of four cases holds:

- (a) $q_1^k = q_1^j$, for some j , $1 \leq j < k$; or
- (b) $q_1^k = q_2^j$ for some j , $1 \leq j < k$; or
- (c) $q_2^k = q_1^j$, $1 \leq j \leq k$; or
- (d) $q_2^k = q_2^j$, $1 \leq j < k$.

In each of the these four cases we can establish the required result. We take up here only one case, (say) case (c), i.e., where $q_2^k = q_1^j$.

In general we choose the leftmost “ q_1 ” for q' . In our case, case (c), $q' = q_1^j$. In cases (b) and (d) it would be q_1^{j+1} . Let $d = \gamma_1 \xi_{n+1} \gamma_2 \xi_{n+1} \dots \gamma_j$. In case $j = 1$, $\gamma_1 = d$ will not describe C_{n+1} from q in \mathcal{A}_{q_0} , and therefore we let $\gamma = dc_{n+1}$, where c_{n+1} is defined below.

For i , $1 \leq i \leq n$, $c_i = c_i^j$, $C'_i = C_i^j$; c_i describes C_i in \mathcal{A} from q and C'_i in \mathcal{A}' from q' .

Let $c_{n+1} = \xi_{n+1} \gamma_{j+1} \xi_{n+1} \dots \gamma_k \xi_{n+1} c_n$. By the construction of c_{n+1} it is clear that $\delta'(q', c_{n+1}) = q'$. We define $C'_{n+1} = \overline{\delta'(q', c_{n+1})}$. It is clear that C'_{n+1} contains C'_n , and that c_{n+1} describes C_{n+1} in \mathcal{A} from q (because it contains ξ_{n+1} and the gamma's describe c_n).

By virtue of c_{n+1} describing C_{n+1} from q in \mathcal{A} , γ is as required, and $\delta(q_0, \alpha\gamma) = q$ and $\delta'(q'_0, \alpha\gamma) = q'$. ■

If we consider alternating chains, and equivalent automata, we get the following as corollary to Theorem 2.9.1 (see also [55, 24]):

Theorem 2.9.2 (Alternating Chain Correspondence) *Let $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$ and $\mathcal{A}' = \langle T = (Q', q'_0, \Sigma, \delta'), \phi' \rangle$ be two equivalent DOA, each of whose states is reachable. Given an alternating chain $C_1 \subset C_2 \subset C_3 \subset \dots \subset C_n$ in DOA \mathcal{A} and $q \in C_1$, there exists an alternating chain $C'_1 \subset C'_2 \subset C'_3 \subset \dots \subset C'_n$ in \mathcal{A}' , $\beta \in \Sigma^*$, and $q' \in C'_1$, and words $c_1, c_2, \dots, c_n \in \Sigma^*$ such that for each pair (C_k, C'_k) ,*

$polarity(C_k) = polarity(C'_k)$, and c_k describes C_k from q in \mathcal{A} , and C'_k from q' in \mathcal{A}' , and $\delta(q_0, \beta) = q$ and $\delta'(q'_0, \beta) = q'$.

Proof: By Theorem 2.9.1, the required β is $\alpha\gamma$. For each i , $1 \leq i \leq n$, $\sigma_i = \beta c_i^\omega$ has run with infinity set C_i and C'_i respectively in \mathcal{A} and \mathcal{A}' . Since $L(\mathcal{A}) = L(\mathcal{A}')$, $polarity(C_i) = polarity(C'_i)$. ■

Since a k -pair DRA (DSA) cannot support a positive (negative) chain of index greater than k , it follows from Theorem 2.9.2 that:

Theorem 2.9.3 [55, 24] *Suppose \mathcal{A} is a DOA. The index of the longest positive (negative) chain in \mathcal{A} , the Rabin (Streett) index of \mathcal{A} , is the minimum number of pairs required to realize $L(\mathcal{A})$ as a DRA (DSA).*

Also, the following facts are easily observed:

1. The index of the longest positive and negative chain in a DOA can differ by at most one, i.e., the Rabin Index (RI) and Streett Index of a language can differ by at most one.
2. If $Rn(k)$ ($St(k)$) denotes the set of ω -regular languages that can be realized as DRA (DSA) with fewer than $k + 1$ pairs, $Rn(k) \subseteq St(k + 1)$ and $St(k) \subseteq Rn(k + 1)$.
3. More pairs lets you recognize a larger class of languages, i.e.,
 - (a) $Rn(k) \subset Rn(k + 1)$.
 - (b) $St(k) \subset St(k + 1)$.
 - (c) $Rn(k) \subset St(k + 1)$.
 - (d) $St(k) \subset Rn(k + 1)$.

Thus the number of pairs induces an infinite hierarchy within the class of ω -regular languages. As in the case of DSA in G_δ (see Prop. 2.5.5), realizing either a DSA or DRA by an equivalent DRA or DSA or DCA with a minimum number of pairs requires “expanding” the transition structure. We next describe algorithms to construct equivalent minimum-pair automata.

2.10 Translating to a minimum-pair automaton

We present algorithms to translate a given DRA or DSA into a minimum-pair DRA, DSA, or DCA. We first motivate the spirit of our algorithm informally before formally delving into it.

In Section 2.5 we saw that the superset closed sublanguage of a DRA could be realized on the same transition structure as a DBA, whereas DSA required an h -fold expansion of the transition structure, h being the number of pairs in the DSA. Based on the constructions to isolate the superset closed sublanguage, we may also exclude the superset closed sublanguage, i.e., given a DSA or DRA \mathcal{A} , construct a DSA or DRA \mathcal{A}' with language $L(\mathcal{A}) \setminus \text{supLan}(\mathcal{A})$.

Armed with a method to exclude supLan , the basic idea then is to employ in succession this supLan exclusion construction, along with the syntactic complement between the Streett and Rabin acceptance conditions.

The effect of the algorithm on the polarity of the set of SCSs of the automaton is illustrated by means of an example. Consider the set of SCSs of a DRA shown in the top-left section of Figure 2.7. The Rabin Index of the automaton is 2, since the longest positive chain is of index 2, for instance the chain $4 \subset 5 \subset 6$. After the first complement, the language corresponding to the negative superset closed SCSs is identified, i.e., the set $L_1 = \{9\}$. Then, after a complement, the language corresponding to the SCSs, $U_1 = \{6, 7, 8, 9\}$, is identified, and then to the set of SCSs $L_2 = \{3, 5, 6, 7, 8, 9\}$, and finally to the set of SCSs $U_2 = \{2, 3, 4, 5, 6, 7, 8, 9\}$. The sets form a chain $L_1 \subset U_1 \subset L_2 \subset U_2$. A SCS S , is accepting provided either $S \in (U_1 \setminus L_1)$ or $S \in (U_2 \setminus L_2)$. Since at each step the superset closed sublanguage will be identified as a Buchi-language, i.e., by means of identification of a set of states on a transition structure, as we see below we get a chain automaton.

We now formally describe the construction to translate a DRA into a minimum-pair DCA and prove its correctness.

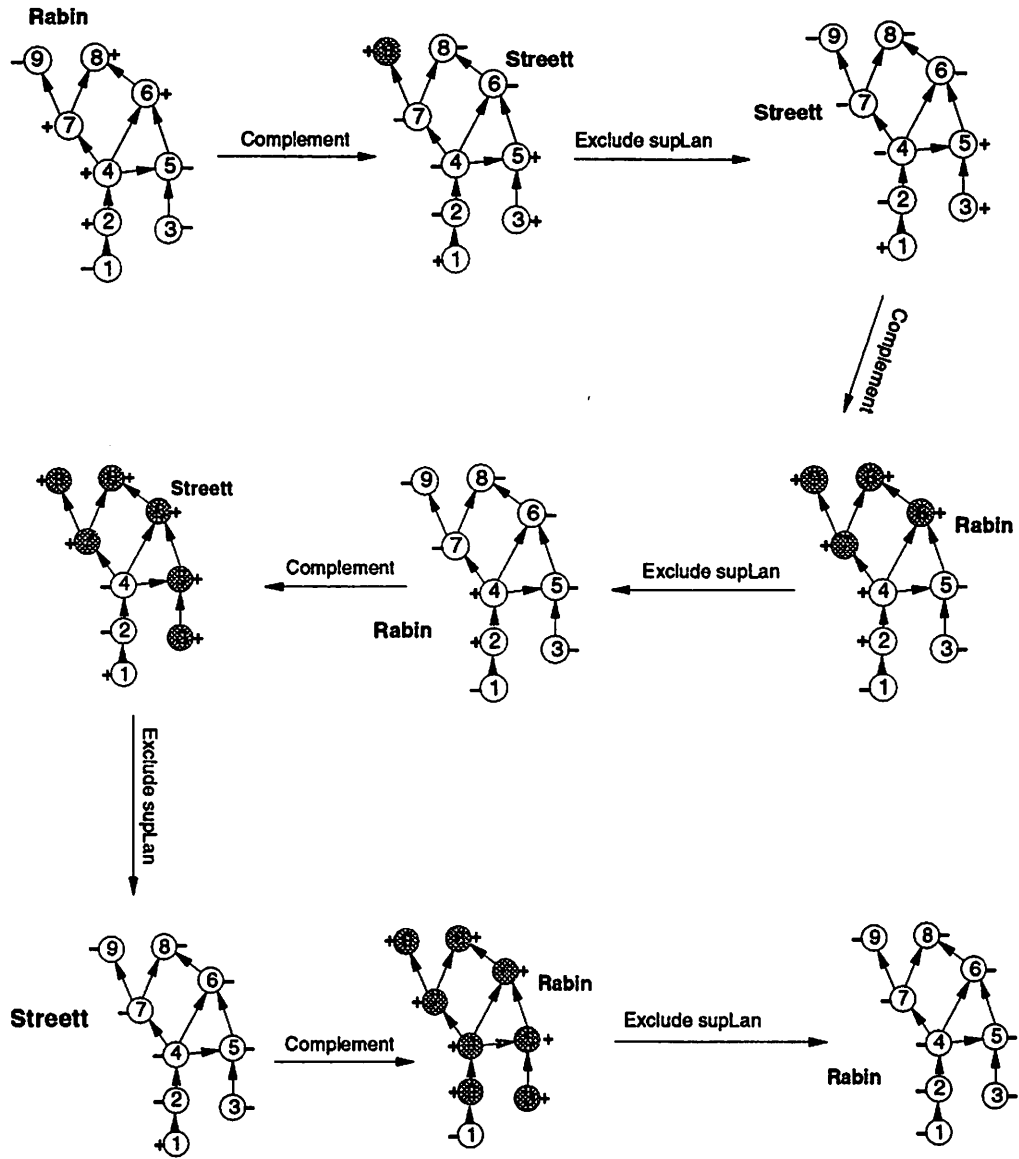


Figure 2.7: DRA2DRCA in action: effect on polarity of SCSs

2.10.1 Excluding $supLan$

In this section, we build on the constructions of Section 2.5 that realize the superset closed sublanguage of both DSA and DRA as DBA, to delete or exclude the $supLan$. We define two constructions, $RabExcl$ and $StrExcl$, that take as input respectively a DRA and a DSA \mathcal{A} , and return respectively:

1. a DRA and DSA, \mathcal{A}' , such that $L(\mathcal{A}') = L(\mathcal{A}) \setminus supLan(\mathcal{A})$, and
2. A state-set, F , being the final state-set for a DBA on the transition structure of \mathcal{A}' and realizing language $supLan(\mathcal{A})$.

RABIN EXCLUSION CONSTRUCTION ($RabExcl$)

INPUT: DRA $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi = \bigvee_{i=1}^h (L_i \wedge \neg(\overline{U_i})) \rangle$.

OUTPUT: DRA $\mathcal{A}' = \langle T, \phi' = \bigvee_{i=1}^h ((L_i \setminus F) \wedge \neg(\overline{U_i \setminus F})) \rangle$, and F , the set of final states of \mathcal{A} .

Lemma 2.10.1 *RabExcl has the property that $L(\mathcal{A}') = L(\mathcal{A}) \setminus supLan(\mathcal{A})$, and $L(\langle T, F \rangle) = supLan(\mathcal{A})$, where $\langle T, F \rangle$ is a DBA with final states F .*

Proof: Theorem 2.5.4 gives us $L(\langle T, F \rangle) = supLan(\mathcal{A})$.

$L(\mathcal{A}) \setminus supLan(\mathcal{A}) = L(\langle T, \phi \wedge \neg F \rangle)$. The Boolean formula $\phi \wedge \neg F$
 $\bigvee_{i=1}^h (L_i \wedge \neg \overline{U_i}) \wedge \neg F \Leftrightarrow \bigvee_{i=1}^h (L_i \wedge \neg \overline{U_i} \wedge \neg \overline{F}) \Leftrightarrow \bigvee_{i=1}^h (L_i \wedge \neg \overline{U_i} \wedge \overline{F}) \Leftrightarrow \bigvee_{i=1}^h (L_i \wedge \neg \overline{U_i \setminus F})$
 $\Leftrightarrow \bigvee_{i=1}^h ((L_i \setminus F) \wedge \neg \overline{U_i \setminus F})$ where the last equivalence follows because a Rabin formula $L_i \wedge \neg \overline{U_i}$ is equivalent to $(L_i \setminus \overline{U_i}) \wedge \neg \overline{U_i}$. ■

Note that we did not need an extra pair to exclude the $supLan$ from a DRA, and we used the same transition structure. However, since the Streett condition is conjunctive, excluding the $supLan$ from a DSA requires adding an additional pair (to make the language smaller) as well as modifying the transition structure.

STREETT EXCLUSION CONSTRUCTION ($StrExcl$)

INPUT: DSA $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \bigwedge_{i=1}^h (L_i \vee \neg(\overline{U_i})) \rangle$

OUTPUT: DSA $\mathcal{A}' = \langle T' = (Q', q'_0, \Sigma, \delta'), \bigwedge_{i=1}^{h+1} (L'_i \vee \neg(\overline{U'_i})) \rangle$ and $F \subseteq Q'$, where if F_i is set of final states of $\mathcal{A}_i = \langle T = (Q, q_0, \Sigma, \delta), L_i \vee \neg(\overline{U_i}) \rangle$

$Q' = Q \times \{1, \dots, h\}$, $q'_0 = (q_0, 1)$

$$\delta'((q, j), a) = \begin{cases} (\delta(q, a), j) & \text{if } q \notin F_j \\ (\delta(q, a), j+1) & \text{if } q \in F_j \text{ and } j < h \\ (\delta(q, a), 1) & \text{if } q \in F_h \end{cases}$$

$$F = \{(q, i) \mid (q, i) \in Q' \text{ and } q \in F_i\},$$

$$L'_i = L_i \times \{1, 2, \dots, h\}, \quad U'_i = U_i \times \{1, 2, \dots, h\}, \text{ and}$$

$$(L'_{h+1}, U'_{h+1}) = (\emptyset, \overline{F}).$$

Lemma 2.10.2 *Let $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \wedge_{i=1}^h (L_i \vee \neg(\overline{U}_i)) \rangle$ and $\mathcal{A}' = \langle T' = (Q', q'_0, \Sigma, \delta'), \wedge_{i=1}^{h+1} (L'_i \vee \neg(\overline{U}'_i)) \rangle$ be as in StrExcl. Then if $\hat{\mathcal{A}} = \langle T', \wedge_{i=1}^h (L'_i \vee \neg(\overline{U}'_i)) \rangle$, $L(\mathcal{A}) = L(\hat{\mathcal{A}})$.*

Proof: Let r_σ be the run of $\sigma \in \Sigma^\omega$ on \mathcal{A} , and r'_σ be the run of σ on $\hat{\mathcal{A}}$. Then, $\Pi_Q(r'_\sigma) = r_\sigma$, and hence $\Pi_Q(\text{inf}(r'_\sigma)) = \text{inf}(r_\sigma)$, and

$$\text{inf}(r_\sigma) \models \wedge_{i=1}^h (L_i \vee \neg(\overline{U}_i)) \Leftrightarrow \text{inf}(r'_\sigma) \models \wedge_{i=1}^h (L'_i \vee \neg(\overline{U}'_i))$$

Thus $L(\mathcal{A}) = L(\hat{\mathcal{A}})$. ■

Lemma 2.10.3 *StrExcl has the property that $L(\mathcal{A}') = L(\mathcal{A}) \setminus \text{supLan}(\mathcal{A})$ and $\text{supLan}(\mathcal{A}) = L(\langle T', F \rangle)$. \mathcal{A}' has $n.h$ states and $h+1$ pairs, where n and h are respectively the number of states and pairs in \mathcal{A} .*

Proof: That $\text{supLan}(\mathcal{A}) = L(\langle T', F \rangle)$ follows from Theorem 2.5.7.

From Lemma 2.10.2 we have $L(\mathcal{A}) = L(\langle T' = (Q', q'_0, \Sigma, \delta'), \wedge_{i=1}^h (L'_i \vee \neg(\overline{U}'_i)) \rangle)$. Since a SCS C can satisfy Streett pair $(\emptyset, \overline{F})$ only if $C \cap F = \emptyset$, $L(\mathcal{A}') = L(\mathcal{A}) \cap \overline{\text{supLan}(\mathcal{A})}$; $L(\mathcal{A}') = L(\mathcal{A}) \setminus \text{supLan}(\mathcal{A})$. ■

The next two lemmas assert that if we exclude the *supLan*, complement, and again exclude the *supLan*, the state-set identifying the second language excluded will include the state-set identifying the first language excluded.

Lemma 2.10.4 *Suppose $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi = \vee_{i=1}^h (L_i \wedge \neg(\overline{U}_i)) \rangle$ is a DRA. Let $(\mathcal{A}', F) = \text{RabExcl}(\mathcal{A})$, and $(\mathcal{A}'', E) = \text{StrExcl}(\overline{\mathcal{A}'})$. Then $F \times \{1, 2, \dots, h\} \subseteq E$.*

Proof: Each Rabin-pair (L_i, U_i) of \mathcal{A} gets transformed to Rabin-pair $(L_i \setminus F, U_i \setminus F)$ of \mathcal{A}' , and corresponding to this pair, in DSA $\overline{\mathcal{A}'}$ we have Streett-pair $(L'_i, U'_i) = (Q \setminus (L_i \setminus F), Q \setminus (U_i \setminus F))$. Thus, since $L'_i \subseteq F$, for each i , $1 \leq i \leq h$, the final state-set of $\langle T, L'_i \vee \neg \overline{U'_i} \rangle$ contains F , and hence $F \times \{1, 2, \dots, h\} \subseteq E$. ■

Similarly:

Lemma 2.10.5 *Suppose $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi = \bigwedge_{i=1}^h (L_i \vee \neg(\overline{U_i})) \rangle$ is a DSA. Let $(\mathcal{A}', E) = \text{StrExcl}(\mathcal{A})$, and $(\mathcal{A}'', F) = \text{RabExcl}(\overline{\mathcal{A}'})$. Then $E \subseteq F$.*

2.10.2 DRA to DRCA construction

We now apply the constructions *RabExcl* and *StrExcl* to devise a translation from a DRA to an equivalent minimum-pair DRCA.

Algorithm: DRA2DRCA

INPUT: DRA $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \bigvee_{i=1}^h (L_i \wedge \neg(\overline{U_i})) \rangle$

OUTPUT: DRCA \mathcal{A}'

Begin

$i = 0; \mathcal{A}_0 = \mathcal{A}$

Repeat

$i = i + 1;$

1. $\mathcal{A}_{i1} = \overline{\mathcal{A}_{i-1}}$

2. $(\mathcal{A}_{i2}, E_i) = \text{StrExcl}(\mathcal{A}_{i1})$

3. $\mathcal{A}_{i3} = \overline{\mathcal{A}_{i2}}$

4. $(\mathcal{A}_i = \langle T_i = (Q_i, q_{0_i}, \Sigma, \delta_i), \phi_i \rangle, F_i) = \text{RabExcl}(\mathcal{A}_{i3})$

until $L(\mathcal{A}_i) = \emptyset$

Return DRCA $\mathcal{A}' = \langle (Q_k, q_{0_k}, \Sigma, \delta_k), \bigvee_{i=1}^k (\neg \pi_{ki}^{-1}(E_i) \wedge \pi_{ki}^{-1}(F_i)) \rangle$,

where k is the RI of $L(\mathcal{A})$.

End

Each iteration of the algorithm, DRA2DRCA, consists of four steps:

1. complementing the DRA \mathcal{A}_{i-1} to yield DSA \mathcal{A}_{i1} .
2. Excluding the *supLan* from DSA \mathcal{A}_{i1} to get DSA \mathcal{A}_{i2} .
3. Complementing DSA \mathcal{A}_{i2} to get DRA \mathcal{A}_{i3} .
4. Excluding the *supLan* from DRA \mathcal{A}_{i3} to get DRA \mathcal{A}_i .

Step 2 is the only operation that expands the transition structure. If \mathcal{A}_{i1} has h pairs then *StrExcl* expands the transition structure h -fold. Therefore the state space of \mathcal{A}_1, Q_1 , is $Q \times \{1, 2, \dots, h\}$. Although *StrExcl* adds a pair, as we show in Lemma 2.10.6, \mathcal{A}_i has exactly h -pairs. Therefore $Q_i = Q_{i-1} \times \{1, \dots, h\}$, and for $0 \leq i < j \leq k$, $Q_j = Q_i \times \{1, 2, \dots, h\}^{j-i}$.

Notation: $\pi_{ji} : Q_j \rightarrow Q_i$, when $j > i$ is the natural projection map; $\pi_{ji}^{-1} : Q_i \rightarrow Q_j$ is the inverse projection map: $\pi_{ji}^{-1}(q_i) = \{q_j | \pi_{ji}(q_j) = q_i\}$.

We characterize DRA2DRCA and establish its complexity through a sequence of lemmas.

Lemma 2.10.6 *In each iteration of DRA2DRCA, the pair added in step 2 is redundant after step 4.*

Proof: *StrExcl* excludes the superset closed sublanguage at step 2, adding a pair $(\emptyset, \overline{E_i})$. On complementing, this becomes a Rabin pair (E_i, Q_i) . Lemma 2.10.5, implies that the state set returned at step 4, F_i , contains E_i , i.e., $F_i \supseteq E_i$. In *RabExcl* F_i is deleted from each set in each pair, transforming the added Rabin pair (E_i, Q_i) to $(\emptyset, Q_i \setminus F_i)$ —a Rabin pair with empty language that can be deleted. ■

Hence at the end of each iteration we have exactly h pairs in \mathcal{A}_i .

Notation: $C = A \sqcup B$, is defined to be $A \cup B$ if and only if $A \cap B = \emptyset$.

Lemma 2.10.7 *In DRA2DRCA, $L(\mathcal{A}_{i-1}) = L(\mathcal{A}_i) \sqcup L(\langle T_i, (\neg E_i \wedge F_i) \rangle)$.*

Proof: $L(\mathcal{A}_{i3}) = L(\mathcal{A}_{i-1}) \sqcup \text{supLan}(\overline{\mathcal{A}_{i-1}})$. Now, $\text{supLan}(\overline{\mathcal{A}_{i-1}}) = L(\langle T_i, E_i \rangle)$, and $L(\mathcal{A}_{i3}) = L(\mathcal{A}_i) \sqcup L(\langle T_i, F_i \rangle)$. Therefore, $L(\mathcal{A}_{i-1}) = L(\mathcal{A}_i) \sqcup (L(\langle T_i, F_i \rangle) \setminus L(\langle T_i, E_i \rangle))$. ■

Lemma 2.10.8 *In DRA2DRCA, if $L(\mathcal{A}_i) \neq \emptyset$, then the Rabin Index of \mathcal{A}_i is one less than the Rabin Index of \mathcal{A}_{i-1} , i.e., $RI(\mathcal{A}_i) = RI(\mathcal{A}_{i-1}) - 1$.*

Proof: From Lemma 2.10.7, $L(\mathcal{A}_{i-1}) = L(\langle T_i, \phi_i \vee (\neg E_i \wedge F_i) \rangle)$, and thus the RI of \mathcal{A}_{i-1} equals the RI of $\mathcal{B} = \langle T_i, \phi_i \vee (\neg E_i \wedge F_i) \rangle$. Note also that $L(\mathcal{A}_i) \cap L(\langle T_i, F_i \rangle) = \emptyset$.

Consider a maximal positive chain in \mathcal{B} . It either ends in a positive (“+”) SCS or a negative (“-”) SCS. If it ends in “+”, this SCS intersects F_i and its polarity is reversed in \mathcal{A}_i , and the polarity of all other SCSs in the chain remain unaltered.

If the maximal positive chain ends in “-”, let the last two sets in the chain be C_1 and C_2 ; $C_1 \subset C_2$, polarity(C_1) = +, and polarity(C_2) = -. It follows that C_2 intersects E_i and C_1 intersects $F_i \setminus E_i$, and thus in \mathcal{A}_i both SCSs will have polarity -; the polarity of the other SCSs in the chain will remain unaltered since they do not intersect F_i . Thus the index of the chain has decreased by one in \mathcal{A}_i .

Every maximal positive chain in \mathcal{B} gets transformed to a positive chain of index one less in \mathcal{A}_i . ■

Theorem 2.10.9 *Let $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \bigvee_{i=1}^h (L_i \wedge \neg(\overline{U}_i)) \rangle$ and $\mathcal{A}' = \langle (Q_k, q_{0_k}, \Sigma, \delta_k), \bigvee_{i=1}^k (\neg\pi_{ki}^{-1}(E_i) \wedge \pi_{ki}^{-1}(F_i)) \rangle$ be as in DRA2DRCA. Then $L(\mathcal{A}) = L(\mathcal{A}')$, and \mathcal{A}' is a DRCA with nh^k states and k pairs, where $n = |Q|$ is the number of states of \mathcal{A} , h is the number of pairs, and k is the Rabin Index of \mathcal{A} . DRA2DRCA has time complexity $O(|T|h^{k+2})$.*

Proof: Since each iteration reduces the index of each maximal positive chain by 1, we have k —the RI of the language—iterations, before $L(\mathcal{A}_k) = \emptyset$; at this stage ϕ_k is unsatisfiable.

Lemma 2.10.2 and 2.10.7, imply that for each $i, 1 \leq i \leq k$,

$$L(\mathcal{A}) = L(\langle T_i, \phi_i \vee \bigvee_{j=1}^i (\neg\pi_{ij}^{-1}(E_j) \wedge \pi_{ij}^{-1}(F_j)) \rangle) \quad (2.1)$$

Since ϕ_k is unsatisfiable, we have $L(\mathcal{A}) = L(\langle T_k, \bigvee_{j=1}^k (\neg\pi_{kj}^{-1}(E_j) \wedge \pi_{kj}^{-1}(F_j)) \rangle)$.

From Lemma 2.10.5, for each $i, 1 \leq i \leq k$, $E_i \subseteq F_i$; and, from Lemma 2.10.4, for each $i, 1 \leq i \leq (k-1)$, $\pi_{(i+1)i}^{-1}(F_i) \subseteq E_{i+1}$. Together with Lemma 2.10.2 we deduce

that the sets comprising the pairs of \mathcal{A}' form a chain, i.e.,

$$\pi_{k1}^{-1}(E_1) \subset \pi_{k1}^{-1}(F_1) \subset \dots \subset \pi_{kk}^{-1}(E_k) \subset \pi_{kk}^{-1}(F_k) \quad (2.2)$$

Thus \mathcal{A}' is indeed a deterministic chain automaton.

Now we address the number of states in \mathcal{A}' . Lemma 2.10.6 asserts that in each iteration \mathcal{A}_{i1} has exactly h -pairs. Therefore step 2, the only step where the transition structure is expanded, expands the number of states h -fold in each iteration, i.e., $|Q_i| = |Q_{i-1}| * h$. Therefore $|Q_k| = nh^k$.

Time complexity: From the nature of *StrExcl*, i.e., because it “copies” the transition structure, it follows that $|T_k| = |T|h^k$. The non-trivial steps of the k^{th} iteration of DRA2DRCA: step 2, step 4, and the stopping test, take the following time:

Step 2: $O(|T_k|h) = O(|T|h^{k+1})$.

Step 4: Since, $F_k \supset E_k$ (Lemma 2.10.4), all transitions “leaving a copy” will be from states in E_k . Therefore $F_k \setminus E_k$ can be computed in each copy separately, taking $O(h^k|T|h^2)$ time, h^k being the number of copies, and $|T|(h+1)^2 = O(|T|h^2)$ being the the time it takes to compute $F_k \setminus E_k$ on one copy. Thus step 4 takes $O(h^{k+2}|T|)$ time.

Stopping test: Checking if $L(\mathcal{A}_k) = \emptyset$ takes $O(|T_k|h)$ time, i.e., $O(|T|h^{k+1})$ time.

Thus the k^{th} iterations takes $O(|T|h^{k+2})$ time. Since the k^{th} iteration is the computationally dominant iteration, DRA2DRCA takes $O(|T|h^{k+2})$ time. ■

DRA2DRCA produces an equivalent DRCA with $n \cdot h^k$ states and k pairs. This is also an equivalent DRA with a minimum—Rabin Index—number of pairs. Since a Rabin chain formula may also be written as a Streett Chain formula, we also have an equivalent DSA with the same number of states and $k+1$ pairs.

The difference between the Streett and Rabin Indices could be at most one, i.e., $|SI - RI| \leq 1$. Therefore the number of pairs in the DSA, $k+1$, could be at most two more than the Streett Index—the minimum number of Streett pairs required to realize the language. We show in Section 2.11 that the number of pairs in a DCA can be minimized on the same transition structure, in polynomial time.

Nevertheless, we give a construction slightly different from DRA2DRCA to produce now a deterministic Streett chain automaton, with nh^l states, where l is the Streett Index of the language.

Algorithm: DRA2DSCA

INPUT: DRA $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \bigvee_{i=1}^h (L_i \wedge \neg(\overline{U}_i)) \rangle$

OUTPUT: DSCA \mathcal{A}'

Begin

$i = 0$; $\mathcal{A}_0 = \mathcal{A} = \langle T_0 = (Q_0, q_0, \Sigma, \delta), \bigvee_{i=1}^h (L_i \wedge \neg(\overline{U}_i)) \rangle$

Repeat

$i = i + 1$;

1. $(\mathcal{A}_{i1}, F_1) = RabExcl(\mathcal{A}_{i-1})$

2. $\mathcal{A}_{i2} = \overline{\mathcal{A}_{i1}}$

3. $(\mathcal{A}_{i3}, E_1) = StrExcl(\mathcal{A}_{i2})$

4. $\mathcal{A}_i = \langle T_i = (Q_i, q_0, \Sigma, \delta_i), \phi_i \rangle = \overline{\mathcal{A}_{i3}}$

until $L(\mathcal{A}_{i3}) = \emptyset$

Return DCA $\mathcal{A}' = \langle (Q_l, q_0, \Sigma, \delta_l), \bigwedge_{i=1}^l (\neg\pi_{li}^{-1}(E_i) \vee \pi_{l(i-1)}^{-1}(F_i)) \rangle$,

where l is the SI of $L(\mathcal{A})$.

End

The difference between DRA2DSCA and DRA2DRCA is the reordering of the steps. Also note that in DRA2DSCA the termination test is $L(\mathcal{A}_{i3}) = \emptyset$. The spirit of DRA2DSCA is the same as that of DRA2DRCA. Each iteration reduces the Streett Index by one.

Theorem 2.10.10 *Let $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \bigvee_{i=1}^h (L_i \wedge \neg(\overline{U}_i)) \rangle$ and $\mathcal{A}' = \langle (Q_k, q_0, \Sigma, \delta_k), \bigwedge_{i=1}^l (\neg\pi_{li}^{-1}(E_i) \vee \pi_{l(i-1)}^{-1}(F_i)) \rangle$ be as in DRA2DSCA. Then $L(\mathcal{A}) = L(\mathcal{A}')$, and \mathcal{A}' is a DSCA with nh^l states and l pairs, where $n = |Q|$ is the number of states of \mathcal{A} , h is the number of pairs, and l is the Streett Index of \mathcal{A} . DRA2DSCA has time complexity $O(|T|h^{l+2})$.*

2.11 Characterizing deterministic Chain automata

We characterize DOA that can be realized as DCA on the same transition structure.

Theorem 2.11.1 *Let $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$ be a DMA. Then there exists a DCA $\mathcal{A}' = \langle T, \phi \rangle$ such that $L(\mathcal{A}) = L(\mathcal{A}')$ if and only if \mathcal{A} is positive and negative union closed.*

Proof: “ \Rightarrow .” If there exists an equivalent DCA \mathcal{A}' on the same transition structure as \mathcal{A} then \mathcal{A} is certainly both positive and negative union closed.

“ \Leftarrow .” By Theorem 2.5.3 the superset closed sublanguage of negative union closed DOA is the language identified by the DBA with the final state set the final states of the DOA.

Since $\overline{\mathcal{A}}$ is also positive and negative union closed, let $E_1 \subseteq Q$ capture the positive superset closed SCSs, N_1 , of $\overline{\mathcal{A}}$, i.e., the negative superset closed SCSs of \mathcal{A} . Exclude N_1 from the set of acceptance sets of $\overline{\mathcal{A}}$, i.e., add E_1 to the set of acceptance sets of \mathcal{A} to get DMA \mathcal{A}'_1 . It is easy to see that \mathcal{A}'_1 is also positive and negative union closed. Let F_1 capture the positive superset closed SCSs of \mathcal{A}'_1 , P_1 . Clearly $N_1 \subseteq P_1$, and $E_1 \subseteq F_1$. Now delete P_1 from the set of acceptance sets of \mathcal{A}'_1 to get DMA \mathcal{A}_1 . The Rabin Index of \mathcal{A}_1 , $\text{RI}(\mathcal{A}_1) = \text{RI}(\mathcal{A}) - 1$, and $L(\mathcal{A}) = L(\mathcal{A}_1) \cup L(\langle T, \neg E_1 \wedge F_1 \rangle)$. Repeating the process on \mathcal{A}_1 and so on, yields a Rabin chain: $E_1 \subset F_1 \subset E_2 \subset F_2 \subset \dots \subset E_k \subset F_k$, where k is the RI of \mathcal{A} , such that $L(\mathcal{A}) = L(\langle T, \bigvee_{i=1}^k (\neg E_i \wedge F_i) \rangle)$. ■

Thus a DOA can be realized as a DRCA or DSCA on the same transition structure iff it is both positive and negative union closed.

Corollary 2.11.2 *Let $\mathcal{A} = \langle T, \phi \rangle$ be a DSA that is negative union closed, or a DRA that is positive union closed. Then there is an equivalent DCA $\mathcal{A}' = \langle T, \phi' \rangle$.*

Proof: The Streett (Rabin) condition is *a priori* positive (negative) union closed and so Theorem 2.11.1 is applicable. ■

Applying Corollary 2.11.2 to the algorithms DRA2DRCA and DRA2DSCA yields algorithms to minimize the number of pairs, in deterministic Chain automata without altering the transition structure:

Theorem 2.11.3 *Let $\mathcal{A} = \langle T, \phi \rangle$ be a DCA (DRCA or DSCA). \mathcal{A} can be translated into a minimum-pair DRCA or DSCA on the same transition structure as \mathcal{A} in time $O(k|T|h)$ and $O(l|T|h)$ respectively, where $k = RI(\mathcal{A})$ and $l = SI(\mathcal{A})$.*

Therefore the RI or SI of a DCA can be determined in polynomial time. How about DMA, DSA, and DRA?

2.12 Computational complexity of determining RI/SI

2.12.1 DMA

The Rabin and Streett Indices for a DMA can be determined in polynomial time as we detail below.

Consider the Haase diagram and the valuation function Val defined in Section 2.7.1. For determining the RI, we compute the valuation function Val , by initializing the value of the top node, Q , to 1 if it is an acceptance set, and 0 otherwise. The Rabin Index of the DMA is $\max_{\{v:v \text{ is a minimal SCS}\}} Val(v)$.

For computing the SI, the value of the top node is initialized to zero. The Streett Index is the value of the bottom node (\emptyset).

2.12.2 DSA & DRA

In this section we consider the complexity of determining the Rabin and Streett Index of DSA and DRA. By Theorem 2.11.3, the RI and SI of DCA can be determined in polynomial time.

Given a DSA with h pairs we can check in polynomial time if the SI or RI is $\leq c$ (for c a constant): if DRA2DRCA (DRA2DSCA) terminates within C iterations, the SI (RI) is $\leq c$.

2.12.3 Determining the Streett/Rabin Index is NP-hard

We show that determining the Streett Index of a DSA is NP-hard. It can also be similarly shown that determining the RI of a DSA is NP-hard.

Consider the following decision problem.

LONGEST NEGATIVE CHAIN (LNC)

INSTANCE: A DSA $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \bigwedge_{i=1}^h (L_i \vee \neg \overline{U}_i) \rangle$ such that the STG induced by δ is strongly connected, and a positive integer $k \leq h$.

QUESTION: Does there exist a negative chain of index $\geq k$?

Theorem 2.12.1 *LNC is NP-complete.*

Proof: LNC is clearly in NP. To show it is NP-hard we will transform the problem MCC, which we define first and then show to be NP-complete.

MAX CONTAINED CYCLE (MCC)

INSTANCE: A strongly connected directed graph $G = (V, E)$, subsets U_1, \dots, U_m of V , and a positive integer $k' \leq m$.

QUESTION: Does there exist a strongly connected subset of the vertices C , such that C is contained in at least k' of the U sets, i.e., $|\{j | C \subseteq U_j\}| \geq k'$?

Lemma 2.12.2 *MCC is NP-complete.*

Proof: MCC is clearly in NP. To show that MCC is NP-hard we transform MIN COVER [18] to it. The construction is due to McMillan [33].

MIN COVER (MC)

INSTANCE: A set $\sigma = \{1, \dots, l\}$, and a set $S = \{S_1, \dots, S_n\}$ of subsets of σ , and a positive integer k .

QUESTION: Does there exist a subset of S of size $\leq k$ whose union is σ ?

Given an instance of MC the corresponding instance of MCC is: a directed graph $G = (V, E)$, where $V \subseteq \{1, 2, \dots, l\} \times \{1, 2, \dots, m\}$ is comprised of l "layers" of vertices:

$$V = \{(i, j) | i \in S_j\}$$

and the edge set $E \subseteq V \times V$ is:

$$E = \{((i, j), (i', j')) | i' = i + 1 \text{ or } i = l \text{ and } i' = 1\}$$

In addition, we create n subsets U_1, \dots, U_n , where $U_j = \overline{R_j}$, and $R_j = \{(i, j) | 1 \leq i \leq l \wedge (i, j) \in V\}$. $k' = n - k$.

Note that the graph G created is strongly connected and there are no self-loops and hence no cycles involving single vertices. Furthermore any strongly connected set has to involve at least one vertex from each layer. Observe that U_j is the set of vertices not in R_j ; therefore, if SCS $C \subseteq U_j$ then " $C \cap S_j = \emptyset$," i.e., for any $i \in \{1, 2, \dots, l\}$, $\bar{A}(i, j) \in C$.

Assume there is a SCS C contained in $\geq k'$ U_j 's. (Say) $C \subseteq U_i$ for each $i \in \{i_1, i_2, \dots, i_r\} \subseteq \{1, 2, \dots, n\}$, where $r \geq k'$. It is easy that $\{S_j : j \notin \{i_1, i_2, \dots, i_r\}\}$ is a cover of σ of size $\leq n - k' = k$.

Conversely, let $\{S_j : j \in J\}$ where $J \subseteq \{1, 2, \dots, n\}$ and $|J| \leq k$ be a cover of σ . Then the simple cycle C formed by picking for each $i \in \{1, 2, \dots, l\}$ some $j \in J$ such that $i \in S_j$ is contained in U_p for $p \notin J$. ■

We now transform MCC to LNC. The transition structure of the DSA, the LNC instance will augment the graph G of MCC, i.e., the STG will contain the graph $G = (V, E)$ as a subgraph. Create a pair $(V \setminus \{s_i\}, \emptyset)$ for each $s_i \in V$, ensuring that no singleton subset of V can be an accepting SCS. For each set U_i of MCC construct a pair (L'_i, U'_i) . Start with $L'_i = \emptyset$ and $U'_i = U_i$. The next part of the construction involves adding states to the pairs (L'_i, U'_i) and adding states/edges to the transition structure. The new states we add will comprise the set V' , and the state-set of the DSA will be $V \sqcup V'$.

For each state $s_i \in V$ such that $|\{j | s_i \in U_j\}| \geq k'$:

1. For each j s.t. $s_i \notin U_j$: create a state s_{ijl} and add edges $(s_i, s_{ijl}), (s_{ijl}, s_i)$. Add s_{ijl} to L'_j and to U'_q for every q s.t. $s_i \in U'_q$.
2. For each j s.t. $s_i \in U_j$: create two states s_{ij1} and s_{ij2} . Add edges $(s_i, s_{ij1}), (s_{ij1}, s_i), (s_i, s_{ij2}),$ and (s_{ij2}, s_i) . Add s_{ij1} and s_{ij2} to all U'_m s.t. $s_i \in U'_m$ and $m \neq j$. Also add s_{ij2} to L'_j .

Mark any state of V as initial state. Augment the alphabet suitably to result in a deterministic structure. k for the LNC instance is $(k' + 1)$.

The transformation is clearly computable in polynomial time. We prove next that the MCC instance has a SCS that is contained in $\geq k'$ U 's iff the DSA created has a negative chain of index $\geq (k' + 1)$.

\Rightarrow : Assume a SCS C such that $C \subseteq U_j$ for at least k' distinct j . Since C cannot be a singleton $C \cap (V \setminus \{s_i\}) \neq \emptyset$ for each $s_i \in V$. Also C cannot be contained in all the subsets U since the NP-hard instance of MC has to have at least one set in the cover. Thus $C \not\subseteq U'_j$ for some j , and therefore C is a rejecting SCS (call it B_1) in DSA \mathcal{A} . For each j such that $C \not\subseteq U_j$, $\exists s_q \in C$ such that $s_q \notin U_j$. Use states s_{qj1} to construct accepting SCS G_1 . Now every state $s_q \in C$ is in $\geq k'$ U 's. Use s_{qj1} and s_{qj2} states for j such that $s_q \in U_j$ to alternately create rejecting and accepting sets to increase the chain index by k' .

\Leftarrow : first note that in any negative chain, G_1 has to have at least two states from V . Furthermore by virtue of all additional states in the LNC instance being added as "loops" on states of V it follows that any accepting SCS of the DSA projected to V is strongly connected in V as well. Therefore G_1 projected to V , call it C' , is strongly connected and since the index of the chain is $\geq (k' + 1)$ it follows that $C \subseteq U'_i$ for at least k' such U 's. But since $C' \subseteq V$ and $C' \subseteq U'_i$, it follows that $C' \subseteq U_i$ for k' such U_i 's. ■

Remark: Although, in the transformation above, the alphabet is a function of the MCC instance, we can modify it to result in an LNC instance with an alphabet of size 2, still resulting in a polynomial transformation.

2.13 Lower bounds for translation in $G_{\delta\sigma} \cap F_{\sigma\delta}$

In this section we assess the upper bounds established by our constructions of the previous sections vis-a-vis other constructions in the literature, and also consider lower bounds.

We presented constructions to translate a DRA on n states and h pairs, into either a DRCA or DRSA with nh^m states and k and l pairs respectively, where k is the RI of the language, l is the SI, and $m = \min(k, l)$. We use Safra's notation and represent the transformations as $DR(n, h) \rightarrow DRC(n2^{m \log h}, k)$ and $DR(n, h) \rightarrow DSC(n2^{m \log h}, l)$.

The previous best construction was Safra's [45] achieving the transformation $DR(n, h) \rightarrow DS(n2^{h \log h}, h + 1)$. Our construction is superior when $m < h$, and is asymptotically no worse; it is to be noted that $k \leq \min(h, \lceil \frac{n}{2} \rceil)$. There is also another construction, $DR(n, h) \rightarrow DRC(n^h 4^{2h^2})$ [8].

Algorithms to translate DRA into DSA [45, 26] have been adapted to yield DCA [15, 27] on the same transition structure as the DSA. The question arises as to why this was possible.

The constructions of [45, 26] were memory-based in the following sense.

Let $\mathcal{A}_1 = \langle T_1 = (Q_1, q_0, \Sigma, \delta_1), \phi_1 \rangle$ and $\mathcal{A}_2 = \langle T_2 = (Q_2, q_0, \Sigma, \delta_2), \phi_2 \rangle$ be two equivalent DOA. If $Q_2 = Q_1 \times M$, and $\forall (q_1, m) \in Q_2$, and $\sigma \in \Sigma$, $\delta_2((q_1, m), \sigma) = (q'_1, m')$ with $q'_1 = \delta_1(q_1, \sigma)$, we say \mathcal{A}_2 is \mathcal{A}_1 with memory, i.e., the Q_1 part of δ_2 matches δ_1 .

If \mathcal{A}_1 is a DRA and \mathcal{A}_2 is a DSA, and \mathcal{A}_2 is \mathcal{A}_1 with memory, then \mathcal{A}_2 which is a priori positive union closed is also negative union closed: let S be a SCS in \mathcal{A}_2 . Since \mathcal{A}_1 and \mathcal{A}_2 are equivalent, the polarity of S in \mathcal{A}_2 has to be the same as the polarity of $\Pi_{Q_1}(S)$ in \mathcal{A}_1 . Therefore DSA \mathcal{A}_2 is also negative union closed. Thus as a corollary to Theorem 2.11.1 we have:

Corollary 2.13.1 *Suppose \mathcal{A}_1 and \mathcal{A}_2 are two equivalent DOA. If \mathcal{A}_1 is a DSA (DRA) and \mathcal{A}_2 is \mathcal{A}_1 with memory, and \mathcal{A}_2 is a DRA (DSA), then \mathcal{A}_2 may be converted into an equivalent DCA on the same transition structure.*

2.13.1 Lower bounds for translating DSA & DRA

Theorems 2.6.3 and 2.6.4 already establish an exponential lower bound on the number of states of the equivalent DBA for a DSA or DMA in G_δ . Furthermore, this establishes the essential optimality of our translations from DSA and DMA in G_δ into equivalent DBA. However, for DSA in G_δ there is always an equivalent DBA of size polynomial in the size of the DSA.

For general DSA, i.e., in $G_{\delta\sigma} \cap F_{\sigma\delta}$, with n states and h pairs our construction results in an equivalent DCA with $n2^{k \log h}$ states, where k the RI of the language, is less than or equal to $\min(\lceil \frac{n}{2} \rceil, h)$.

Safra and Vardi [44], have shown that for every $n \geq 2$, there exists a DSA on $3n$ states and $2n$ pairs such that every equivalent NBA has at least 2^n states. Since for every n -state h -pair DRA there is an equivalent NBA with $n(h+1)$ states, it follows that in the equivalent DRA for the DSA of [44] either the number of states or number of pairs is exponential in n . There is still a gap of $\log h$ in the exponent between the inferred lower bound from [44] and our upper bound.

We show a better lower bound next that bridges this gap.

Theorem 2.13.2 *For every $n \geq 2$, there exists a DSA (DRA) with $3n$ states and $2n$ pairs such that any equivalent DRA (DSA) has at least $2^n n!$ states.*

Proof: We use the same language-family as in [44]. The alphabet $\Sigma = \{0, a, b\}$. A word $\sigma \in \Sigma^\omega$ is viewed as an infinite sequence of vectors each of size n . The language L_n is all sequences where for all i , $1 \leq i \leq n$, a appears infinitely many times in position i iff b does too. That is, for a sequence $\sigma \in \Sigma^\omega$ consider:

$$\text{inf}_a(\sigma) = \{i \in [1 \dots n] : |\{j : j \bmod n = i, \sigma(j) = a\}| = \infty\}$$

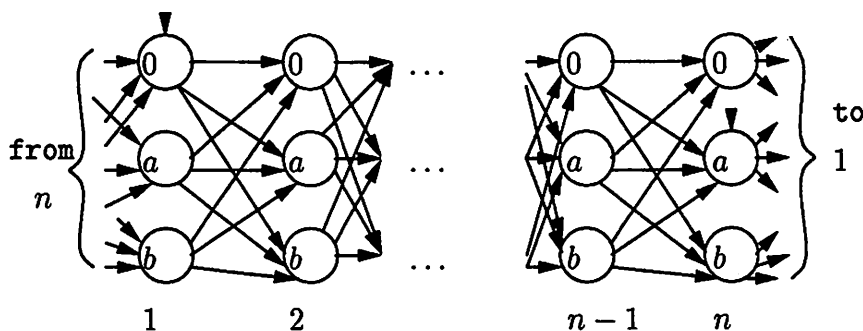
and:

$$\text{inf}_b(\sigma) = \{i \in [1 \dots n] : |\{j : j \bmod n = i, \sigma(j) = b\}| = \infty\}$$

$\sigma \in L_n$ iff $\text{inf}_a(\sigma) = \text{inf}_b(\sigma)$.

L_n is accepted by the DSA $\mathcal{A}_n = \langle (Q_n, q_{0_n}, \{0, a, b\}, \delta_n), \bigwedge_{i=1}^n (\{(i, a)\} \Rightarrow \{(i, b)\}) \wedge (\{(i, b)\} \Rightarrow \{(i, a)\}) \rangle$. The state-set $Q_n = \{1, 2, \dots, n\} \times \{0, a, b\}$, the initial state $q_{0_n} = (1, 0)$, the alphabet is $\{0, a, b\}$, and the transition function, δ_n , induces the STG shown in Figure 2.8; the label on the arc should be the label of the destination state. There are $2n$ pairs: a run is accepting provided at each level the a -state is visited infinitely often iff the b -state is visited infinitely often.

Consider a maximal positive chain in \mathcal{A}_n . A maximal positive chain is $\mathcal{C} = C_1 \subset C_2 \subset \dots \subset C_{2n+1}$, where $C_1 = \{1, 2, \dots, n\} \times \{0\}$, $C_{2n+1} = \{1, 2, \dots, n\} \times \{0, a, b\}$, and there is a sequence $I_1 \subset I_2 \subset \dots \subset I_n$, where for j , $1 \leq j \leq n$, $I_j \subseteq \{1, 2, \dots, n\}$, and C_{2j} either is $C_{2j-1} \cup (I_j \setminus I_{j-1}) \times \{a\}$, or $C_{2j-1} \cup (I_j \setminus I_{j-1}) \times \{b\}$, and $C_{2j+1} = C_{2j-1} \cup (I_j \setminus I_{j-1}) \times \{a, b\}$. C_i is positive for odd i . The RI of \mathcal{A}_n is $n+1$.

Figure 2.8: DSA \mathcal{A}_n accepting L_n

Given such a maximal positive chain \mathcal{C} , and $q = q_{0_n}$, by Theorem 2.9.2 (Alternating Chain Correspondence), in any equivalent DRA \mathcal{A}'_n there exists a corresponding chain positive chain $\mathcal{C}' = \mathcal{C}'_1 \subset \mathcal{C}'_2 \subset \dots \subset \mathcal{C}'_{2n+1}$, and state $q'_\mathcal{C}$, and words $c_1, \dots, c_{2n+1} \in \{0, a, b\}^*$ such that c_i describes \mathcal{C}_i from $q = q_{0_n}$ in \mathcal{A}_n and \mathcal{C}'_i from $q'_\mathcal{C}$ in \mathcal{A}'_n .

Let $\mathcal{C} = \mathcal{C}_1 \subset \mathcal{C}_2 \subset \dots \subset \mathcal{C}_{2n+1}$ and $\mathcal{K} = \mathcal{K}_1 \subset \mathcal{K}_2 \subset \dots \subset \mathcal{K}_{2n+1}$ be two distinct maximal positive chains in DSA \mathcal{A}_n . For the corresponding chains \mathcal{C}' and \mathcal{K}' in an equivalent DRA \mathcal{A}_n , the claim is that $q'_\mathcal{C} \neq q'_\mathcal{K}$.

Assume not for contradiction, i.e., let $q'_\mathcal{C} = q'_\mathcal{K} = q'$. Let $2p$ be the first index where \mathcal{C} and \mathcal{K} are different, $\mathcal{C}_{2p} \neq \mathcal{K}_{2p}$; the first index where the two chains differ has necessarily to be even. As was mentioned earlier there is a sequence of subsets of $\{1, 2, \dots, n\}$, I , corresponding to a maximal positive chain in \mathcal{A}_n . Let this sequence be $I_1 \subset I_2 \subset \dots \subset I_n$ for chain \mathcal{C} , and $J_1 \subset J_2 \subset \dots \subset J_n$ for \mathcal{K} . Let $I_p \setminus I_{p-1} = \{i\}$ and $J_p \setminus J_{p-1} = \{j\}$. There are two cases to consider now:

1. $I_p = J_p$. In this case, $j = i$, and either $(j, a) \in \mathcal{C}_{2p}$ and $(j, b) \in \mathcal{K}_{2p}$, or $(j, b) \in \mathcal{C}_{2p}$ and $(j, a) \in \mathcal{K}_{2p}$. Either way, consider the run of word $\sigma = (c_{2p}k_{2p})^\omega$ from $q' \in \mathcal{A}'_n$ and q in \mathcal{A}_n . Its infinity set will be $\mathcal{C}_{2p} \cup \mathcal{K}_{2p}$ in \mathcal{A}_n , a positive SCS, and in \mathcal{A}'_n it will be $\mathcal{C}'_{2p} \cup \mathcal{K}'_{2p}$, a negative set (DRA are negative union closed)—a contradiction.
2. $I_p \neq J_p$. Let q be the least index (necessarily greater than p) such that $j \in I_q$, and r the least index (again necessarily $> p$) such that $i \in J_r$. Now consider

word $\sigma = (c_{2q}k_{2r})^\omega$. Its run in \mathcal{A}_n from q_0 has infinity set $C_{2q} \cup K_{2r}$ a positive SCS, whereas the run of σ in \mathcal{A}'_n from q' has infinity set $C'_{2q} \cup K'_{2r}$, a negative SCS being the union of two negative SCSs—contradiction.

Therefore there is at least one distinct state in an equivalent DRA for every distinct maximal positive chain in DSA \mathcal{A}_n . It is easy to see that there are $2n(2n-2)(2n-4)\dots 1$, i.e., $2^n n!$, distinct maximal positive chains in \mathcal{A}_n . ■

The upper bound established by our construction would be $3n(2n)^{n+1}$ which is $2^{O(n \log n)}$, as is the lower bound. Thus we have established a tight lower bound for the translation of DSA into DRA.

It is interesting to note that for the same family of languages the lower bound [44] for translation into NBA is 2^n , and the lower bound for translation into DRA is $2^n n!$. Both lower bounds are asymptotically tight; there is a translation $DS(n, h) \rightarrow NB(nh2^h)$.

2.13.2 Lower bound for translating DMA

There is an elegant translation from a DMA into a DCA based on the *latest visitation record* (LVR) [20, 34, 53]. The LVR is a permutation of the set of states, Q , of the automaton. If $|Q| = n$ the LVR, R , may be thought of as a vector $R[1, 2, \dots, n]$ that is associated with each state of a run over Q , where $R[n]$ is the most recently visited state and $R[1]$ is the least recently visited state in the run. If R is the LVR at the current state $R[n]$ of the run, and $q' = R[j]$ is the next state in the run, the new LVR at q' , R' , is defined to be $R'[1, 2, \dots, n] = R[1 \dots (j-1)]R[(j+1) \dots n]R[j]$, i.e., we move the current state to the “front” of the LVR.

To transform a DMA into a DRA, in addition to the LVR we maintain a hit value h , $1 \leq h \leq |Q|$, that records the index in the previous LVR of the current state; for instance, in R and R' considered in the previous paragraph, the hit value for R' is j . Thus, corresponding to a run we have a sequence of LVRs and hit values.

The equivalent DCA \mathcal{A}' for a DMA $\mathcal{A} = \langle (Q, q_0, \Sigma, \delta), \bigvee_{F \in \mathcal{F}} (\bigwedge_{f \in F} (f) \wedge_{q \notin F} (\neg q)) \rangle$, is defined over state space, $Q' = \text{Perm}(Q) \times \{1, 2, \dots, n\}$, where $\text{Perm}(Q)$ is the set of permutations of Q and $|Q| = n$, and the second component is the hit value. Let

the initial state be (P, n) where P is some permutation of Q such that $P[n] = q_0$ and the initial hit value is n . Given a current LVR and hit value and a letter from Σ , the next LVR and hit value are determined as discussed earlier. The chain condition, with a chain of length $2n$, $E_1 \subseteq F_1 \subseteq E_2 \subseteq F_2 \subseteq \dots \subseteq E_n \subseteq F_n$, arises as follows:

- E_i consists of states in Q' with hit value greater than $n + 1 - i$.
- $F_i = E_i \cup \{(R, n+1-i) \mid (R, n+1-i) \in Q' \text{ and } \cup_{j=n}^{n+1-i} R[j] \text{ is an acceptance set}\}$.

It is clear that the $E_i \subseteq F_i$ and $F_i \subseteq E_{i+1}$, and that DCA \mathcal{A}' is equivalent to the DMA \mathcal{A} .

The equivalent DCA produced may have up to $n!n$ states. We show an $n!$ lower bound.

Definition 2.13.1 Consider the power-set lattice (also called the subset lattice) on a finite set $\{1, 2, \dots, n\}$. Level i in this lattice refers to all sets of cardinality i , and we say level i is marked $+$ to denote that every set in level i is a positive set. In the alternating power-set lattice level n is marked $+$, level $n - 1$ is marked $-$, level $n - 2$ is marked $+$, and so on with every alternate level marked $+$.

Theorem 2.13.3 For every $n \geq 2$, there exists a DMA on n states such that any equivalent DCA has at least $n!$ states.

Proof: For every $n > 2$, consider the DMA $\mathcal{A}_n = \langle T^n, \phi_n \rangle$, where T^n is defined in Definition 2.6.2 and ϕ_n denotes the positive acceptance sets of the powerset lattice.

Let $\mathcal{C} = C_1 \subset C_2 \subset \dots \subset C_n$, and $\mathcal{K} = K_1 \subset K_2 \subset \dots \subset K_n$ be two maximal chains in \mathcal{A}_n . Every maximal chain in \mathcal{A}_n is alternating and has the same index.

Let \mathcal{A}'_n be an equivalent DCA. By Theorem 2.9.2, there are chains \mathcal{C}' and \mathcal{K}' in \mathcal{A}'_n corresponding to \mathcal{C} and \mathcal{K} in \mathcal{A}_n .

Since Q_n , the state set of T^n , is of size n , $C_n = K_n$ and has polarity $+$. Assume that $C_{n-1} \neq K_{n-1}$. Consider C'_{n-1} and K'_{n-1} in \mathcal{A}'_n . Even if $C'_n \cap K'_n \neq \emptyset$, the claim is that $C'_{n-1} \cap K'_{n-1} = \emptyset$. Assume otherwise for contradiction. By Theorem 2.9.2 and the nature of T^n , there exists $c \in \Sigma_n^*$ and $d \in \Sigma_n^*$ describing respectively C'_{n-1} and K'_{n-1} in \mathcal{A}'_n , and C_{n-1} and K_{n-1} in \mathcal{A}_n from appropriate states, such that $\text{inf}(c^w) = C_{n-1}$,

and $\text{inf}(d^\omega) = K_{n-1}$. By Lemma 2.6.2, there is an $f \in \Sigma_n^*$ such that f describes $C'_{n-1} \cup K'_{n-1}$ from some state in $C'_{n-1} \cap K'_{n-1}$ such that $\text{inf}(f^\omega) = \text{inf}(c^\omega) \cup \text{inf}(d^\omega)$. f^ω will have the run in \mathcal{A}_n with infinity set $\text{inf}(f^\omega) = C_n = K_n$, and thus $f^\omega \in L(\mathcal{A}_n)$. \mathcal{A}'_n being claimed equivalent to \mathcal{A}_n implies $C'_{n-1} \cup K'_{n-1}$ should have polarity $+$. But since DCA are negative union closed $C'_{n-1} \cup K'_{n-1}$ will have polarity $-$, a contradiction.

Now assume that $C_{n-1} = K_{n-1}$, but $C_{n-2} = K_{n-2}$. The claim is that even if $C'_n \cap K'_n \neq \emptyset$, and $C'_{n-1} \cap K'_{n-1} \neq \emptyset$, $C'_{n-2} \cap K'_{n-2} = \emptyset$. The argument is similar to the $n - 1$ case coupled with the observation that the Chain condition is also positive union closed.

Therefore, if i is the greatest index where chains \mathcal{C} and \mathcal{K} differ, then $C'_i \cap K'_i = \emptyset$. Also $C'_{i+1} \setminus C'_i \geq 1$, there is at least one state unique to C'_{i+1} (at “level” $i+1$). Since there are n distinct subsets of $Q_n = \{1, 2, \dots, n\}$ of size $n - 1$, we have at least n states at level $n - 1$, and for each of these sets of size $n - 1$ there are $n - 1$ distinct subsets of size $n - 2$, and so on.

Therefore the number of states in DCA \mathcal{A}'_n is at least:

$$1 + n + \frac{n^2}{2} + \dots + \frac{n!}{3!} + \frac{n!}{2!} + n!$$

which is clearly greater than $n!$. ■

We can similarly derive lower bounds for the translation of DMA into DSA and DRA.

Theorem 2.13.4 *For every even $n \geq 2$, there exists a DMA on n states such that any equivalent DRA (DSA) has at least $n(n - 2)(n - 4)\dots 1$ states, which is $2^{\frac{n}{2}}(n/2)!$.*

2.14 Minimization of DOA

Thus far we have addressed the problem of simplifying the form and size of the acceptance condition. In this section, we consider the problem of minimizing the number of states in a DOA. Given a DOA, we are interested in computing the equivalent DOA with the fewest number of states among all equivalent DOA.

This problem is well characterized for DFAs, i.e., deterministic *-automata, by the Myhill-Nerode theorem [22] which guarantees that every *-regular language is

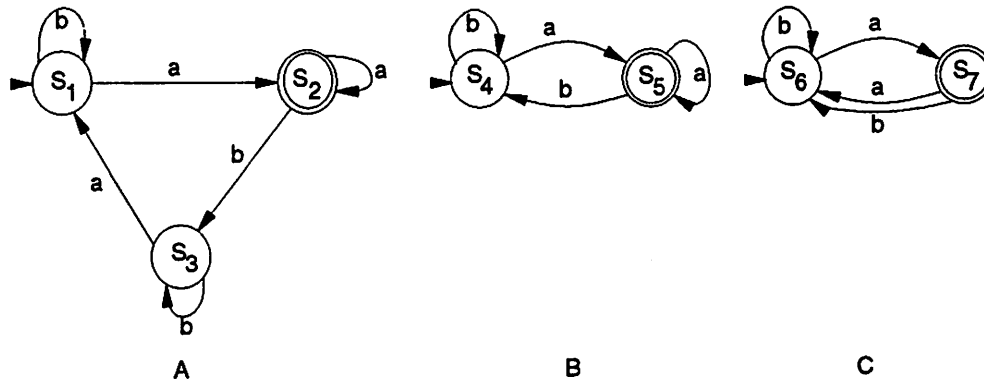


Figure 2.9: Counter-examples to DBA minimization possibilities

represented by a unique (up to isomorphism) minimum state DFA, and thus can be computed in time polynomial in the size of the (unminimized) automaton.

A closed or open ω -regular language is essentially a $*$ -regular language. Given a DOA representing an open or closed ω -regular language, the transformations of Section 2.4 define equivalent DBA that denote a special and unique $*$ -regular language. For a closed language we get a unique prefix closed $*$ -language, and for an open language we get a unique $*$ -language L such that if a word $\alpha \in L$ then $\alpha\beta \in L$, where β is any word. These DFA can be minimized to yield a unique minimum DFA and viewed as DBA they are also the unique minimum-state DBA for the corresponding ω -regular language.

Unfortunately, even for DBA there is not a unique minimum-state DBA. In Figure 2.9, DBAs B and C are two clearly non-isomorphic 2-state DBA accepting sequences over a and b with infinite occurrences of a 's (minimum?—a one state automaton can only accept the empty language or Σ^ω).

Since the language of a DBA is of the form $limW$ where W is the $*$ -language of the “underlying” DFA, minimization of the DFA is a sound procedure to (possibly) reduce the number of states of the DBA. Automaton A in Figure 2.9, is an instance of a minimum state DFA but a non-minimum state DBA; the minimum-state DBA is either B or C .

A certificate for minimality of DFAs is the non-existence of equivalent states. This

is however not the case for DBA, as automata B and C in Figure 2.9 exemplify—all states are equivalent, i.e., generate the same language.

Staiger [49] showed that the language equivalence based procedure, i.e., that of “collapsing” equivalent states *à la* DFA-minimization, works for a subclass of DBA-realizable languages, the class $G_\delta \cap F_\sigma$.

Definition 2.14.1 *Given $L \subseteq \Sigma^\omega$, and $w \in \Sigma^*$ we define the left quotient L/w of L by the word w as $L/w = \{b : b \in \Sigma^\omega \text{ and } w.b \in L\}$.*

Definition 2.14.2 *Suppose $L \subseteq \Sigma^\omega$. The transition structure associated with L is the deterministic automaton on alphabet Σ , state set $\{L' \mid L' = L/w \text{ for some } w \in \Sigma^*\}$, initial state L , and transition function δ with $\delta(L, \sigma) = L/\{\sigma\}$.*

Trachtenbrot considered a class of ω -languages (not necessarily regular) that were defined by finite state associated transition structures. Staiger [49] showed that every ω -language with a finite-state associated transition structure in the class $G_\delta \cap F_\sigma$ is regular, and is accepted by a DMA on the associated transition structure.

We give a simple proof of Staiger’s theorem next.

Theorem 2.14.1 (Staiger [49]) *Let $\mathcal{A} = \langle (Q, q_0, \Sigma, \delta), \phi \rangle$ be a DOA such that $L(\mathcal{A}) \in G_\delta \cap F_\sigma$, and $T' = (Q', q'_0, \Sigma, \delta')$ be the associated transition structure for $L(\mathcal{A})$. Then for some $F' \subseteq Q'$, the DBA $\langle T', F' \rangle$ is equivalent to \mathcal{A} .*

Proof: We first show that if $\sigma_1, \sigma_2 \in \Sigma^\omega$ are two sequences such that $\sigma_1 \in L(\mathcal{A})$ and $\sigma_2 \notin L(\mathcal{A})$, the infinity sets of the runs of these, r'_1 and r'_2 respectively, on the associated transition structure, T' , cannot intersect, i.e., $\text{inf}(r'_1) \cap \text{inf}(r'_2) = \emptyset$.

Assume for contradiction that $q' \in \text{inf}(r'_1) \cap \text{inf}(r'_2) \neq \emptyset$. Let r_1 and r_2 be the runs respectively of σ_1 and σ_2 on \mathcal{A} . From some index, $I > 0$, all the states occurring in r_1, r_2, r'_1 , and r'_2 are in their respective infinity sets. Let $i, j > I$ be indices such that $r'_1[i] = q'$ and $r'_2[j] = q'$. Let $\sigma'_1 = \sigma_1[i]\sigma_1[i+1]\dots$, the suffix of σ_1 starting at i , and $\sigma'_2 = \sigma_2[j]\sigma_2[j+1]\dots$. Let $\delta(q_0, \sigma_1[0\dots(i-1)]) = q_1^1$ and $\delta(q_0, \sigma_2[0\dots(j-1)]) = q_2^1$.

i and j have been chosen so that the language generated from q_1^1 , $L(\langle (Q, q_1^1, \Sigma, \delta), \phi \rangle)$, equals the language generated from q_2^1 , $L(\langle (Q, q_2^1, \Sigma, \delta), \phi \rangle)$, equals q' (recall states

in T' denote languages). $\sigma'_1 \in q'$ and $\sigma'_2 \notin q'$. From q_1^1 we will alternately apply σ'_2 and σ'_1 to arrive at a contradiction.

Since $\sigma_1 \in L(\mathcal{A})$, q_1^1 will be in a SCC comprised entirely of final states (because $L(\mathcal{A})$ is in $G_\delta \cap F_\sigma$). Therefore the run of σ'_2 from q_1^1 , r_1^1 , has to “descend” to a negative SCC, and let $q_1^2 \in \text{inf}(r_1^1)$ be such that language generated from q_1^2 is q' (such q_1^2 has to exist since application of either σ'_1 or σ'_2 from q' causes repeated visits to q').

Now applying σ'_1 from q_1^2 necessitates another descent into a positive SCC. Proceeding in this manner, alternately applying σ'_2 and σ'_1 requires an infinite number of descents in the DAG of SCCs of \mathcal{A} , clearly impossible because Q is finite.

Therefore, the infinity sets of accepted and rejected sequences cannot intersect in T' . Therefore each SCC is comprised of SCSs of the same polarity. The polarity of a SCC C is + if $L(\langle T', C \rangle) \subseteq L(\mathcal{A})$, where $\langle T', C \rangle$ is the DBA with final states C . ■

Therefore a DOA on n states in $G_\delta \cap F_\sigma$ can be minimized in polynomial time to yield an equivalent minimum-state DBA or Co-DBA with a unique transition structure, requiring $O(n^2)$ language equivalence tests to construct the associated transition structure (T') and $O(n)$ language inclusion tests to determine the polarity of the SCCs of T' .

Since DOA equivalence checking for DMA, DSA, and DRA, is in polynomial time, it follows that the state minimization problem for these automata is in NP, but it is not known if the problem is NP-hard or in P. For DOA not in $G_\delta \cap F_\sigma$, we know of no better procedure than to enumerate all DOA of smaller size to find the minimum-state equivalent DOA. Characterizing the minimum state DOA remains an open problem.

2.15 Determinization

Until now we have mostly considered deterministic ω -automata except for Section 2.4, where we showed that any nondeterministic ω -regular language that is open or closed can be realized as a NBA on essentially the same transition structure. While determining the Rabin Index of a DMA is in polynomial time, and for DSA and DRA it is in polynomial time to check if the RI is less than a constant c and NP-hard in

general, it is not difficult to see (it entails solving language universality) that even determining if a NOA is an open language, is PSPACE-hard.

Furthermore, restricting the number of Rabin and Streett pairs in nondeterministic ω -automata does not induce a hierarchy on the ω -regular languages, as every ω -regular language is accepted by some nondeterministic Buchi automaton. Nondeterministic automata can be much more succinct than deterministic automata, and translating between nondeterministic automata is often not as expensive. An interesting and relevant question here is if the complexity of the determinization of a nondeterministic ω -automaton is a function of the Rabin Index of the language.

Let $\mathcal{A} = \langle (Q, q_0, \Sigma, \delta), F \rangle$ be a nondeterministic Buchi automaton. We are interested in computing the deterministic Rabin automaton, $\mathcal{B} = \langle (Q', q'_0, \Sigma, \delta'), \phi' \rangle$ equivalent to \mathcal{A} . The transformations of Section 2.4 yield special $*$ -languages for closed and open ω -regular languages that can be determinized, by the classical subset construction [41], to yield an equivalent DBA with at most 2^n states, where n is the number of states in \mathcal{A} .

Safra's determinization [42] construction yields a transformation $NB(n) \rightarrow DR(2^{O(n \log n)}, n)$. There is a gap of a factor of $\log n$ in the exponent between the number of states in the deterministic automaton for closed and open languages and Safra's construction. If $L(\mathcal{A}) \in G_\delta$, then the equivalent DRA can be realized as a DBA on the same transition structure; the number of pairs needed is one. This fact, along with our construction $DR(n, h) \rightarrow DRC(n2^{k \log h}, k)$, makes us wonder if perhaps we could achieve $NB(n) \rightarrow DR(2^{O(n \log(k+1))}, k)$, where k is the RI of $L(\mathcal{A})$, i.e., if the complexity of determinization is a function of the RI.

Unfortunately, we observe that there is a family of NBA, where for an $O(n)$ state NBA every equivalent DRA has at least $n!$, i.e., $O(2^{n \log n})$ states.

Proposition 2.15.1 *For every $n > 2$, there exists a NBA \mathcal{A}_n on $n + 2$ states such that $L(\mathcal{A}_n) \in G_\delta$, and every equivalent DRA has at least $n!$ states.*

Proof: We consider the same family of languages considered by Michel [35] to prove the $n!$ lower bound and observe that the languages are in G_δ .

For every $n > 2$, consider the alphabet $E_n = \{a_1, a_2, \dots, a_n, s\}$. For a word

$w \in E_n^\omega$ we associate a directed graph $G(w)$ on vertex set $\{a_1, a_2, \dots, a_n\}$, and edges defined as follows. There is an edge (a_i, a_j) in $G(w)$ if and only if $a_i a_j$ appears infinitely often as a subword of w . The language L_n consists of all words over E_n , w , such that $G(w)$ contains a cycle, i.e., $L_n = \{\sigma \mid \sigma \in E_n^\omega \text{ and } G(\sigma) \text{ contains a cycle}\}$. Michel showed that this language is accepted by a NBA on $n + 2$ states and every equivalent DRA has at least $n!$ states.

We argue that the language L_n is in G_δ . For contradiction assume that the equivalent DRA $\mathcal{B}_n = \langle (Q_n, q_{0_n}, \delta_n, \Sigma_n), \phi_n \rangle$ has 2 SCSs C_1 and C_2 such that $C_1 \subset C_2$, and C_1 is positive but C_2 is negative. For $q \in C_1$, let $\alpha \in \Sigma_n^*$ take q_{0_n} to q , i.e., $\delta_n(q_{0_n}, \alpha) = q$. Let $c_1 \in \Sigma_n^*$ describe C_1 from q and $c_2 \in \Sigma_n^*$ describe C_2 from q . Since C_1 is positive $\sigma_1 = \alpha c_1^\omega \in L_n$ and therefore $G(\sigma_1)$, the graph on the vertex set $\{a_1, a_2, \dots, a_n\}$, will have a cycle. Now consider $\sigma_2 = \alpha(c_1 c_1 c_2 c_2)^\omega$. If $a_i a_j$ appears infinitely often in σ_1 then it appears infinitely often in σ_2 also. Therefore $G(\sigma_1)$ is a subgraph of $G(\sigma_2)$, and hence $G(\sigma_2)$ contains a cycle also, implying that $\sigma_2 \in L_n$. But C_2 is negative, a contradiction. ■

Therefore the factor of $\log n$ blow-up in the exponent is an artifact of considering ω -languages (“passing to infinity”) versus $*$ -languages, and is not a function of the RI of the ω -language.

2.16 Minimum witnesses for non-empty ω -automata

In this section we consider the minimum witness problem (MWP) for ω -automata. Given an ω -automaton $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$, if $L(\mathcal{A}) \neq \emptyset$, we are interested in finding $\sigma, c \in \Sigma^*$ such that σ leads from q_0 to q , i.e., $q \in \delta(q_0, \sigma)$, and c describes a positive SCS C from q , and such that (σ, c) is a minimal witness in the following sense. If W is the set of witnesses to the non-emptiness of $L(\mathcal{A})$, $W = \{(\sigma', c') \mid \sigma', c' \in \Sigma^* \text{ and } q' \in \delta(q_0, \sigma'), \text{ and } c' \text{ describes positive SCS } C' \text{ from } q'\}$, (σ, c) is a minimal witness in W , i.e., $(\sigma, c) = \arg \min_{(\sigma', c') \in W} \{|\sigma'| + |c'|\}$.

The verification question—does a system satisfy a property—can often be trans-

lated to a language emptiness question. The MWP arises thus in returning an “error trace”, i.e., a witness to the non-emptiness of an ω -automaton, to the designer on a failed verification “run” [21]. The returned witness is also used to modify an abstraction of a system in a verification scheme based on verification by iteratively refining an abstraction [2] of the system.

Although, testing for language non-emptiness for Buchi, Rabin, Streett, and Muller automata is in polynomial time [16], we observe in this section that the complexity of the MWP varies depending on the form of the acceptance condition. It is in P for Buchi and Rabin automata, but NP-complete for Streett and Muller automata.

The MWP may be posed as a decision problem as follows:

INSTANCE: $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$, and a positive integer K .

QUESTION: Does there exist $\sigma, c \in \Sigma^*$ such that for some $q \in \delta(q_0, \sigma)$, c describes a positive SCS C from q , and $|\sigma| + |c| < K$.

If $L(\mathcal{A}) \neq \emptyset$, then there is some witness (σ, c) such that $|\sigma| + |c| = O(|Q|^2)$ and therefore $MWP \in NP$. MWP for Streett and Muller can encode the Directed Hamiltonian Circuit Problem (DHCP). The DHCP, is to decide if a given directed graph has a simple cycle or circuit that includes each vertex, and is NP-complete [22]. An instance of DHCP, $G = (V, E)$, is transformed into an instance of the MWP with Streett or Muller condition as follows: $Q = V$, q_0 can be any state in Q , $\Sigma = \{0, 1\}$, δ denotes edges of E labeled with 0 or 1 arbitrarily. ϕ as a Muller formula denotes one acceptance set Q , and as a Streett formula, the pairs $\{(\{q\}, \phi) | q \in Q\}$; a SCS has to contain each $q \in Q$ to satisfy ϕ . K for the MWP instance is $|Q|$. The MWP instance has a positive SCS described by $c \in \{0, 1\}^{|Q|}$ if and only if the DHCP instance admits a Hamiltonian circuit.

For each $q \in Q$ if we could determine the minimally described positive SCS containing q , S_q , then the required minimum witness would be (σ, c) , such that $q_{min} \in \delta(q_0, \sigma)$, and c is the shortest word that describes $S_{q_{min}}$ from:

$$q_{min} = \arg \min_{q \in Q} \{|\pi(q_0, q)| + |S_q|\} \quad (2.3)$$

where $\pi(q_0, q)$ denotes the shortest path from q_0 to q , and allowing ourselves a little abuse of notation $|S_q|$ denotes the length of the smallest word describing S_q . There-

fore, if we could compute S_q efficiently we could determine the minimum witness. For Streett and Muller automata computing S_q is NP-hard.

For a Buchi automaton, if $q \in F$, S_q is the smallest simple loop containing q ; if $q \notin F$, $S_q = \pi(q, f_{min}) \cup \pi(f_{min}, q)$, where $f_{min} = \arg \min_{f \in F} \{|\pi(q, f)| + |\pi(f, q)|\}$.

For a Rabin automaton, for each pair (L_i, U_i) such that $q \in U_i$, S_{iq} , the minimally described positive SCS containing q and satisfying the pair (L_i, U_i) , is computed treating the subgraph of δ on the state set U_i as a Buchi automaton with final states L_i . $S_q = \arg \min_{\{S_{iq} | q \in U_i\}} \{|S_{iq}|\}$.

Since the Buchi and Rabin conditions are negative union closed, $S_{q_{min}}$, is a simple cycle; in this case, the length of the shortest word describing $S_{q_{min}}$ is indeed $|S_{q_{min}}|$.

2.17 Summary

In this chapter we have studied the relative complexity of the different ω -automata within subclasses of the ω -regular languages. Corresponding to the $*$ -regular languages are the open and closed ω -regular languages, and every NOA representing such a language can be converted into an equivalent NBA or nondeterministic Co-Buchi automaton (NCBA) on essentially the same transition structure.

In the class, $G_\delta \cap F_\sigma$, represented by DOA that are both positive superset and subset closed, every DOA may be converted into an equivalent DBA or DCBA on the same transition structure. Every DOA in $G_\delta \cap F_\sigma$ is accepted by a unique minimum-state DOA. Within the class G_δ —the DBA-realizable languages, i.e., DOA with positive superset closed SCSs—negative union closed DOA, and in particular a DRA can be translated into a DBA on the same transition structure, i.e., $DR(n, h) \rightarrow DB(n)$, and for DSA we have a translation $DS(n, h) \rightarrow DB(nh)$. We presented polynomial time algorithms to decide if a given DOA is in G_δ , and to effect the translation into the equivalent DBA. We also provided instances of DMA and DSA, whose equivalent DBA has to have exponentially more states than the number of states in the DMA or DSA.

For every alternating chain in a DOA there is a corresponding alternating chain in

any equivalent DOA. We presented a translation, $DR(n, h) \rightarrow DRC(nh^m, k)$, where $m = \min(k, l)$, and k (l) is the RI (SI) of the language. Positive and negative union closed DOA can be converted into DCA on the same transition structure. The number of pairs in a DCA can be minimized in polynomial time, and DCA can be translated into other ω -automata on the same transition structure. The RI/SI of a DMA or DCA can be determined in polynomial time, whereas it is NP-hard to determine the RI/SI of a DSA or DRA. We showed a lower bound $DS(3n, 2n) \rightarrow DR(n!2^n, -)$, and also proved that the LVR-based translation of a DMA into a DCA is essentially optimum.

The complexity of determining a NBA is not a function of the RI of the language. While language emptiness for Buchi, Rabin, Streett, and Muller automata is in polynomial time, we can produce a minimum-size certificate for the nonemptiness of a Buchi or Rabin automaton in polynomial time, whereas it is NP-hard to do the same for Streett or Muller automata.

In the next chapter we build on the results of this chapter to formulate algorithms to decide the winner, and derive the winner's strategy, in two-person games of perfect information played on ω -automata.

Chapter 3

Two-person Games of Perfect Information on ω -automata

3.1 Introduction

Consider the following game between two players, player-0 and player-1. In a single-step of this game, player-0 picks a letter from his finite alphabet Σ_0 , and then player-1 picks a letter from her finite alphabet Σ_1 . These single-steps repeat ad infinitum to form a play of the game. Player-0 wins the game if the play is in the game language, $\Gamma \subseteq (\Sigma_0 \times \Sigma_1)^\omega$; otherwise, player-1 wins, i.e., when the play is in $\bar{\Gamma}$.

A strategy for a player is a rule that tells the player what letter to pick at each instant; and, a winning strategy is one that ensures a win for the player no matter how the other player behaves. Each player observes the choice of the other player at each step, and the strategy may be a function of the entire history of the play—a game of perfect information. The game is determined if one of the players has a winning strategy.

This infinite game of perfect information was first considered by Gale and Stewart [17]. They showed that open and closed games, i.e., when Γ is either an open or a closed set, are determined; [59] showed determinacy for G_δ and F_σ games; [11] showed that $G_{\delta\sigma} \cap F_{\sigma\delta}$ games are determined; and, Martin [32] showed that all Borel games are determined.

IF Γ is presented as an ω -automaton, we have a Gale-Stewart game of perfect Information on an ω -automaton. In this case, the winner has a finite-state strategy implementable by a finite state machine (FSM). Buchi and Landweber [7, 40, 20] gave a constructive procedure to solve—decide the winner and synthesize the winner’s strategy FSM—games on ω -automata.

Solving a game on an ω -automaton is also known as Church’s problem [9]. If Γ is a property or synthesis requirement for a synchronous digital circuit to be synthesized, we are interested in knowing if the circuit to be synthesized has a winning strategy against its (adversarial) environment, and if so, we want a FSM implementing a winning strategy subject to certain implementation constraints.

In this chapter we study the complexity of solving two-person games of perfect information on ω -automata. We build on the results of Chapter 2 to devise a synthesis algorithm whose complexity is a function of the structural complexity of Γ .

Section 3.2 sets up games on ω -automata. In Section 3.3 we consider graph games—a variant of games on ω -automata that are convenient for developing decision and synthesis algorithms. Sections 3.4-3.7 contain decision and synthesis algorithms for games of increasing structural complexity in the game language; Section 3.7.3 contains the main result of this chapter. In Section 3.8 we review the correspondence between tree automata and games, and point out the relevance of our results to tree automata. In Section 3.9 we show some lower bounds for the size of strategies required in Muller and Streett games. Section 3.10 summarizes this chapter.

3.2 Game on ω -automaton

The game is played on a deterministic ω -automaton $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$ called the *game automaton*. The game starts at state q_0 ; player-0 picks or *plays* some $\sigma_0^0 \in \Sigma_0$, and then player-1 plays some $\sigma_1^0 \in \Sigma_1$, and the game advances to state $q_1 = \delta(q_0, (\sigma_0^0, \sigma_1^0))$, player-0 then plays some $\sigma_0^1 \in \Sigma_0$, followed by player-1 picking some $\sigma_1^1 \in \Sigma_1$, and so on ad infinitum. The infinite sequence $\sigma = (\sigma_0^0, \sigma_1^0)(\sigma_0^1, \sigma_1^1) \dots$ so constructed forms a *play* of the game; a prefix of σ is a *partial play*. The acceptance condition of the game automaton, ϕ , is called the *winning condition* for player-0. The

type of the game is inherited from the type of ω -automaton the game automaton is, which in turn is determined by the form of the Boolean formula defining the winning condition; for instance, a Buchi game automaton defines a Buchi game also called a G_δ game.

The language of the game automaton, $L(\mathcal{G})$, is called the *game language*. The play $\sigma \in (\Sigma_0 \times \Sigma_1)^\omega$ is a *win for player-0* if $\sigma \in L(\mathcal{G})$, i.e., if the set of states visited infinitely often by the play, $\text{inf}(r_\sigma)$, where r_σ is the run of σ on \mathcal{G} , satisfies the winning condition if $\text{inf}(r_\sigma) \models \phi$. Otherwise, $\text{inf}(r_\sigma) \not\models \phi$, and the play is a win for player-1—her winning condition is $\neg\phi$. In a Buchi game, player-1's winning condition is a Co-Buchi winning condition; in a closed (open) game, player-1's winning condition is an open (closed) winning condition, i.e., the set of winning plays for player-1 forms an open (closed) set.

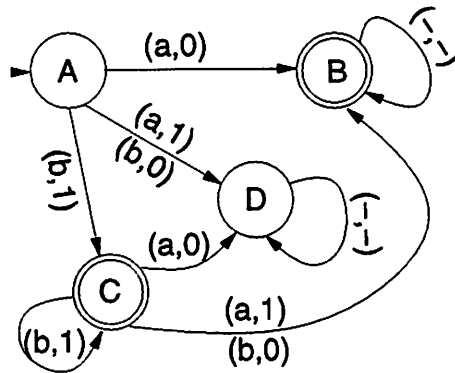


Figure 3.1: Simple Buchi Game

Figure 3.1 shows a simple Buchi game. $\Sigma_0 = \{a, b\}$ and $\Sigma_1 = \{0, 1\}$. $(-, -)$ represents $\Sigma_0 \times \Sigma_1$. The play $(b, 1)(b, 1)(a, 0)^\omega$ is a win for player-1, whereas the play $(b, 1)^\omega$ is a win for player-0.

3.2.1 Players' strategies

A strategy for a player is a rule for picking the next letter based on the play constructed up to the current instant. A *strategy* for player-0 is a function $\psi : \Sigma_1^+ \rightarrow \Sigma_0$, and a strategy for player-1 is a function $\tau : \Sigma_0^+ \rightarrow \Sigma_1$.

Strategy ψ for player-0 is a *winning strategy* if every play that results from him playing according to ψ is a win for him, i.e.,

$$\forall \sigma_1^0 \forall \sigma_1^1 \forall \sigma_1^2 \dots [(\psi(\epsilon), \sigma_1^0) (\psi(\sigma_1^0), \sigma_1^1) (\psi(\sigma_1^0 \sigma_1^1), \sigma_1^2) \dots \in L(\mathcal{G})]$$

Thus no matter how player-1 plays at each instant, player-0 is ensured a win when he plays his winning strategy. Similarly, a strategy τ for player-1 is a winning strategy if by playing according to τ she is guaranteed a win however player-0 plays, i.e.,

$$\forall \sigma_0^0 \forall \sigma_0^1 \forall \sigma_0^2 \dots [(\sigma_0^0, \tau(\sigma_0^0)) (\sigma_0^1, \tau(\sigma_0^0 \sigma_0^1)) (\sigma_0^2, \tau(\sigma_0^0 \sigma_0^1 \sigma_0^2)) \dots \in \overline{L(\mathcal{G})}]$$

A strategy for player-0 can also be thought of as partial function $\psi : (\Sigma_0 \Sigma_1)^* \rightarrow \Sigma_0$ requiring only that it be defined for partial plays that arise from him playing according to ψ . Similarly, player-1's strategy is a partial function $\tau : \Sigma_0 (\Sigma_1 \Sigma_0)^* \rightarrow \Sigma_1$, requiring that it be defined only for partial plays arising from her playing τ .

Clearly at most one of the players can have a winning strategy. A player with a winning strategy is called the *winner* of the game, and is said to *win the game*. A game with a winner is said to be *determined*.

Theorem 3.2.1 (Buchi & Landweber [7]) *Let $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$ be a Gale-Stewart game on a DOA. This game is determined, i.e., either player-0 or player-1 has a winning strategy, and the winner has a regular or finite-state strategy.*

A regular strategy for player-0 is a *Moore* FSM $S_0 = (Q_0, q_0^0, \Sigma_0, \Sigma_1, \delta_0, \lambda_0)$, where:

- Q_0 is the finite state set, and $q_0^0 \in Q_0$ is the initial state
- Σ_0 is the output alphabet, and Σ_1 is the input alphabet
- $\lambda_0 : Q_0 \rightarrow \Sigma_0$ is the *output* function, and
- $\delta_0 : Q_0 \times \Sigma_1 \rightarrow Q_0$ is the transition function.

It is easy to check that FSM S_0 defines a strategy for player-0.

We can associate an ω -regular language $L(S_0) \subseteq (\Sigma_0 \times \Sigma_1)^\omega$ with the strategy FSM S_0 ; $(\sigma_0^0, \sigma_1^0)(\sigma_0^1, \sigma_1^1) \dots \in L(S_0)$, provided $\sigma_0^0 = \lambda_0(q_0^0)$, and for each $i > 0$,

$\sigma_0^i = \lambda_0(\delta_0(q_0^0, \sigma_1^0 \sigma_1^1 \dots \sigma_1^{i-1}))$, where δ_0 is assumed to be inductively extended to $\delta_0 : Q_0 \times \Sigma_1^+ \rightarrow Q_0$. $L(S_0)$ is a closed set. We can thus think of the FSM as an ω -automaton.

Since S_0 and \mathcal{G} are deterministic ω -automata, there exists a unique relation $r \subseteq Q_0 \times Q$ such that:

- $(q_0^0, q_0) \in r$, and
- $(q, q') \in r \Rightarrow \forall \sigma_1 \in \Sigma_1 [(\delta_0(q, \sigma_1), \delta(q', (\lambda_0(q), \sigma_1))) \in r]$

We call such a relation a *tracking relation* [36]. We say state $s \in Q_0$ is *tracked* by the set of states $r(s) \subseteq Q$ in the game automaton and state $q \in r(s)$ *tracks* state s , where $r(s) = \{q \mid (s, q) \in r\}$

$L(S_0)$ represents a winning strategy provided $L(S_0) \subseteq L(\mathcal{G})$. In this case the unique tracking relation is called a *simulation relation* [36], and we say state $s \in Q_0$ is *simulated* by the set of states $r(s) \subseteq Q$ in the game automaton and state $q \in r(s)$ *simulates* s .

Player-1's strategies

Similarly, a regular strategy for player-1 is a *Mealy* FSM $S_1 = (Q_1, q_0^1, \Sigma_1, \Sigma_0, \delta_1, \lambda_1)$, where:

- Q_1 is the finite state set, and $q_0^1 \in Q_1$ is the initial state
- Σ_1 is the output alphabet, and Σ_0 is the input alphabet
- $\lambda_1 : Q_1 \times \Sigma_0 \rightarrow \Sigma_1$ is the output function, and
- $\delta_1 : Q_1 \times \Sigma_0 \rightarrow Q_1$ is the transition function

Notice, the output function λ_1 is a function of the input alphabet and the state, appropriate for representing regular strategies for player-1 who plays second. We can again associate a closed ω -regular language $L(S_1) \subseteq (\Sigma_0 \times \Sigma_1)^\omega$ with the strategy S_1 for player-1; $(\sigma_0^0, \sigma_1^0)(\sigma_0^1, \sigma_1^1) \dots \in L(S_1)$, provided $\sigma_1^0 = \lambda_1(q_0^1, \sigma_0^0)$, and for each $i > 0$, $\sigma_1^i = \lambda_1(\delta_1(q_0^0, \sigma_0^0 \sigma_0^1 \dots \sigma_0^{i-1}), \sigma_0^i)$. S_1 is a winning strategy if $L(S_1) \subseteq \overline{L(\mathcal{G})}$. Since the

game automaton $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$ is a DOA, $\bar{\mathcal{G}} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), -\phi \rangle$ has language $\overline{L(\mathcal{G})}$, and hence if $L(S_1) \subseteq L(\bar{\mathcal{G}})$, there is a unique tracking and simulation relation $r \subseteq Q_1 \times Q$ with the properties:

- $(q_0^1, q_0) \in r$
- $(q, q') \in r \Rightarrow \forall \sigma_0 \in \Sigma_0 [(\delta_1(q, \sigma_0), \delta(q', (\sigma_0, \lambda_1(q, \sigma_0)))) \in r]$

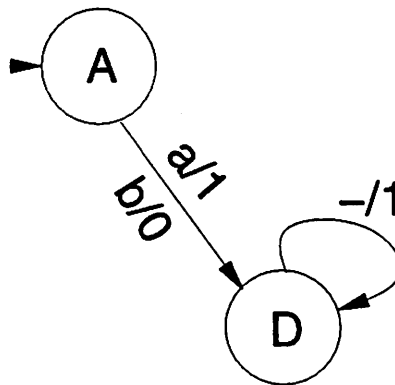


Figure 3.2: Mealy winning strategy for player-1 starting from state A in game of Figure 3.1

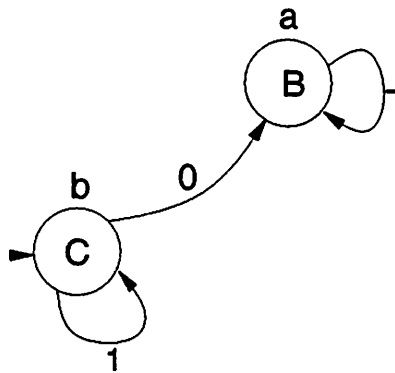


Figure 3.3: Moore winning strategy for player-0 starting from state C in game of Figure 3.1

In the game of Figure 3.1, player-1 has the winning strategy shown in Figure 3.2. In this game, player-1 does not have a winning strategy that is implementable as a Moore FSM.

If we were to consider the game of Figure 3.1, but with the game starting from the state labeled C , player-0's winning strategy is shown in Figure 3.3. The output at a state is the letter written above the state. The simulation relation between the winning strategies and the game automaton, for the winning strategies in both Figure 3.2 and Figure 3.3, is the identity relation on the state labels. The set of states of the game is partitioned into two sets $\{A, D\}$ and $\{C, B\}$, such that player-1 and player-0 respectively win the games starting from states in these sets, i.e., when play starts from any state in these sets. For the game starting from state B , the winning strategy for player-0 is essentially embedded in the winning strategy for the game starting from state C . Why?

Lemma 3.2.2 [34] *Suppose $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$ is a game automaton, and S is the strategy FSM for the winner. If there is a partial play p that arises from the winner playing his/her winning strategy S , and the partial play reaches \hat{q} , i.e., $\delta(q_0, p) = \hat{q}$, then the winner also wins the game starting at state \hat{q} , and his winning strategy FSM is S with initial state being the state the FSM is in at the end of partial play p .*

Amongst the regular strategies we identify a useful special class of strategies.

Memory based strategies

Definition 3.2.1 *Given a game automaton $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$, and a Moore strategy FSM for player-0, $S_0 = (Q_0, q_0^0, \Sigma_0, \Sigma_1, \delta_0, \lambda_0)$, if the unique tracking relation $r \subseteq Q_0 \times Q$ is such that:*

- $\forall q \in Q_0, \exists! q' \in Q$ such that $(q, q') \in r$, then we say strategy S_0 is **memory-based**
- $\forall q' \in Q$, there exists at most one $q \in Q_0$ such that $(q, q') \in r$, we say S_0 is a **memory-less strategy**

In a memory-based winning strategy, a state, $s \in Q_0$, of the strategy FSM is not simulated by more than one state in the game automaton, i.e., $|r(s)| = 1$. Therefore

Q_0 may be expressed as $M \times Q$, where the set M is called the *memory*, and such that:

- $\delta_0 = \delta_m \times \delta_q$, and
- if $\delta_0((m, q), \sigma_1) = (m', q')$, then $q' = \delta(q, (\lambda_0(q, m), \sigma_1))$

The “ q part” of the transition function of the strategy FSM is consistent with the transition function of the game automaton. The *amount of memory* used by the strategy is $|M|$. A memory-less strategy is a memory-based strategy with $M = \emptyset$. Notice that in a memory-less strategy every state in the game automaton tracks at most one state in the strategy FSM.

In the Gale-Stewart game, player-0 and player 1 are not treated alike: Player-0 plays first, and as a result Player-1’s winning strategy may not always be implementable as a Moore FSM. We consider next a related game played on a finite bipartite graph, popularized by McNaughton [34], where both players are treated symmetrically. This game makes for a cleaner development of algorithms to solve Gale-Stewart games on ω -automata, by synthesizing memory-based strategies for the winner.

3.3 McNaughton’s graph game

A (McNaughton) *graph game* is $\mathcal{G} = \langle G, \phi \rangle$. The game is played on a finite directed bipartite graph $G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1)$, called the *game graph*. The game graph G has node-set $N = N_0 \sqcup N_1$ and edge-set $E = E_0 \sqcup E_1$, with $E_0 \subseteq N_0 \times N_1$, $E_1 \subseteq N_1 \times N_0$, and where for each node $n \in N$ there is at least one out-going edge $(n, n') \in E$. The game is played between the players player-0 and player-1. Player-0’s node-set is N_0 and player-1’s N_1 . At any instant of a play of the game the place-marker for the game is on a node $n \in N$; for $i \in \{0, 1\}$, if n is in N_i player- i moves the place-marker along one of the edges $(n, n') \in E_i$ to node n' in N_{1-i} . It is now player- $(1 - i)$ ’s turn to move; and, this repeats ad infinitum.

Figure 3.4 shows a game graph. Player-0’s nodes, $\{B, D, F\}$ are drawn as circles,

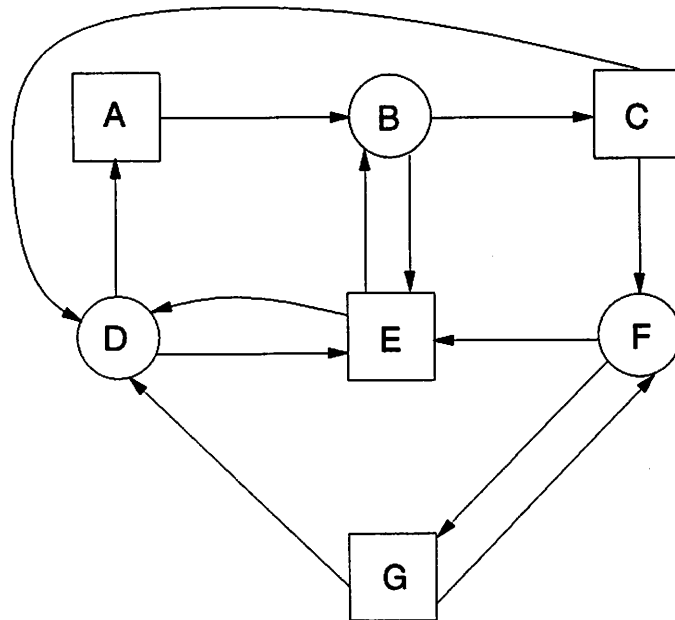


Figure 3.4: A Game Graph

and player-1's nodes $\{A, C, E, G\}$ as squares. Notice that each node has at least one-outgoing edge, and all edges are to the other player's nodes.

We shall find it convenient not to designate a start node for the game. A *play* is an infinite sequence $\eta = n_0 n_1 n_2 \dots \in N^\omega$ such that for each $i \geq 0$, $(n_i, n_{i+1}) \in E$. The winning condition ϕ is defined in a manner analogous to the acceptance condition for ω -automata, as a Boolean formula over the nodes N . The type of the graph game is determined by the type of the winning condition: for instance, a Streett graph game has a Streett formula specifying the winning condition for player-0. A play η is a win for player-0 if $\text{inf}(\eta) \models \phi$; otherwise it is a win for player-1.

Let us use the Streett pairs $(\{D\}, \{B, C, D, E, F, G\})$ and $(\{F\}, \{B, C, D, E, F, G\})$ to specify the winning condition for the game graph of Figure 3.4, to define a Streett graph game. The play $(ABED)^\omega$ is a win for player-1.

3.3.1 Players' strategies

For $i \in \{0, 1\}$, a strategy for player- i is a partial function $\psi : N^* N_i \rightarrow N_{1-i}$, that for each partial play $n_0 n_1 \dots n_k$ ending in N_i gives the edge $(n_k, \psi(n_0 n_1 \dots n_k)) \in E_i$, to move the place-marker along.

Definition 3.3.1 Given a graph-game $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), \phi \rangle$, a regular strategy for player-0 is a game graph $S = \langle G' = (N' = N'_0 \sqcup N'_1, E' = E'_0 \sqcup E'_1), \neg \overline{N'} \rangle$, called the **strategy graph**, such that:

1. there is a function $r : N' \rightarrow N$, where $r : N'_0 \rightarrow N_0$, and $r : N'_1 \rightarrow N_1$, labeling nodes of the strategy graph S with nodes of the game graph, and
2. every node in N'_0 should have exactly one out-going edge and this should match the game graph, i.e., $\forall n'_0 \in N'_0, \exists! n'_1 \in N'_1$ such that
 - (a) $(n'_0, n'_1) \in E'$, and
 - (b) $(r(n'_0), r(n'_1)) \in E$
3. every move of player-1 in the game graph should be tracked in the strategy graph, i.e., $\forall n'_1 \in N'_1$, if $r(n'_1) = n_1$, then for every edge $(n_1, n_0) \in E$, there should be an edge $(n'_1, n'_0) \in E'$ such that $r(n'_0) = n_0$, and
4. every move of player-1 in the strategy graph should correspond to a move of player-1 in the game graph, i.e., let $n'_1 \in N'_1$ and $r(n'_1) = n_1$; for each edge $(n'_1, n'_0) \in E'$, there should be an edge $(n_1, n_0) \in E$ such that $n_0 = r(n'_0)$.

In the strategy graph the moves of player-0 are fixed, and each move of player-1 is tracked. Playing according to strategy S works as follows. Assume the place-marker is at some player-0 node $n_0 \in N_0$, and the strategy place-marker is at node $n'_0 \in N'_0$ such that $r(n'_0) = n_0$. Player-0 moves the place-marker in the strategy graph along the unique edge $(n'_0, n'_1) \in E'$, and the place-marker in the game graph along the edge $(r(n'_0), r(n'_1)) \in E$. If player-1 now moves the place-marker along edge $(r(n'_1), \hat{n}_0) \in E$ in the game graph, the place-marker is moved along the edge $(n'_1, \hat{n}'_0) \in E'$, where $r(\hat{n}'_0) = \hat{n}_0$; this repeats.

The regular strategy is a winning strategy for player-0 if every play that results from him playing according to the strategy is a win for him, i.e., for every play $n'_0 n'_1 n'_2 \dots \in (N')^\omega$, $\text{inf}(r(n'_0)r(n'_1)r(n'_2)\dots) \models \phi$. A winning strategy for player-0, represents a winning strategy for him from the set of nodes of the game graph that occur as labels of nodes in the strategy graph; we also say that player-0 wins the game starting *from this set of nodes*, or the nodes in this set are his *winning nodes*.

A regular strategy for player-0 in a graph game (as in Definition 3.3.1), can be expressed as a game graph on node set $N' = M \times N$, where M is a finite set called the *memory*, with the property that for $(m, n) \in N'$, $r((m, n)) = n$, and $((m, n), (m', n')) \in E'$ only if $(n, n') \in E$. The *amount of memory* necessary to express a regular strategy graph in this form as a *memory-based strategy graph* is

$$|M| = \max_{n \in N} |\{n' | n' \in N' \text{ and } r(n') = n\}|$$

In other words, the amount of memory is the count of the most popular game graph node labeling the strategy graph. If $|M| = 1$ the strategy is termed a *memory-less* strategy. In this case, player-0's strategy is obtained by choosing one edge out of each player-0 node in the game graph that occurs as a label of some node in the winning strategy graph.

Figure 3.5 shows the memory-less winning strategy for player-0 for the Streett graph game of Figure 3.4.

The graph game is perfectly symmetrical; a regular strategy, memory, and winning strategy for player-1 are defined identically for her by interchanging the roles of the players and negating the winning condition.

3.3.2 Translating ω -automaton games into graph games

Given a game automaton $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$ the corresponding graph game is $\mathcal{G}' = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), \phi' \rangle$, where:

- $N_0 = Q_0, N_1 = Q_0 \times \Sigma_0$
- $E_0 = \{(q, (q, \sigma_0)) | q \in Q \text{ and } \sigma_0 \in \Sigma_0\}$

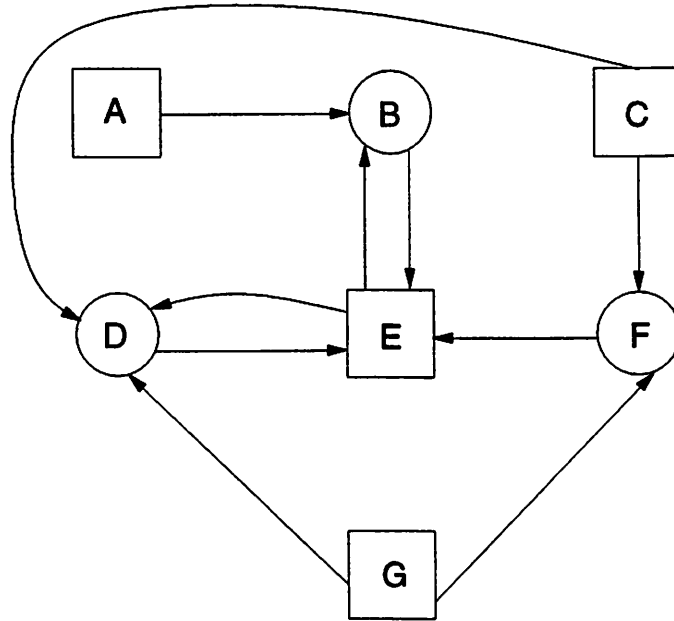


Figure 3.5: Memory-less winning strategy for player-0 in game of Fig. 3.4

- $E_1 = \{((q, \sigma_0), q') \mid q \in Q \text{ and for some } \sigma_1 \in \Sigma_1 \delta(q, (\sigma_0, \sigma_1)) = q'\}$
- ϕ' is defined to be such that for $S \subseteq N$, $S \models \phi' \Leftrightarrow S \cap N_0 \models \phi$. If ϕ is a Rabin or Streett formula, then a pair (L_i, U_i) in ϕ gets translated to pair $(L'_i, U'_i) = (L_i \sqcup (L_i \times \Sigma_0), U_i \sqcup (U_i \times \Sigma_0))$

It is easy to see that there is a one-to-one correspondence between the plays and strategies, between the ω -automaton game and its corresponding graph game, as defined above. The regular strategies of the graph game correspond exactly to the memory-based strategies for the automaton game.

Given a game automaton the size of the corresponding game graph is linear in the size of the automaton.

Sometimes it is useful to think of a game graph as an ω -automaton. Given a graph game $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), \phi \rangle$, let d be the maximum out-degree of any node in N . Let alphabet $\Sigma_d = \{1, 2, \dots, d\}$. For each node $n \in N$, assign the set Σ_d to the edges making sure each edge has at least one symbol. A play is now equivalent to a sequence in Σ_d^ω ; given a sequence in Σ_d^ω starting from a given node, we

have a unique play corresponding to it. If we were to fix the start state of the game (from N_0), we have thus a deterministic ω -automaton whose language is the “set of winning plays” of the original graph game, the game language $\subseteq (\Sigma_d \times \Sigma_d)^\omega$. We can interpret any DOA equivalent to this DOA as a game again (by making two copies of the states and having transitions between the copies giving rise to a bipartite graph), and solve this game to derive winning strategies for our original graph game. When we talk of the topological and structural properties of a graph game, we mean the properties of a DOA as defined above.

Next we consider the question of solving games of increasing complexity in the winning condition. We focus on the Rabin and Streett conditions, and refer the reader to [34] for a good exposition on Muller games. We develop the algorithms on game graphs. We will show that for a given game $\mathcal{G} = \langle G = (N, E), \phi \rangle$, the node set N is partitioned $N = W_0 \sqcup W_1$, where W_0 (W_1) is the set of winning nodes for player-0 (player-1).

3.4 Open and Closed games

Consider a graph-game $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), F \rangle$, with an open language. Recall that in this case player-0 wins a play if it includes a node from F . If W_0 denotes nodes from which player-0 can win the game, clearly $F \subseteq W_0$. For $W \subseteq W_0$, if node n is a player-0 node with an edge into W or a player-1 node with all edges into W , then $n \in W_0$.

Thus, let $W_0^0 = F$. For $i \geq 1$, $W_0^i = W_0^{i-1} \cup W_{00}^i \cup W_{01}^i$, where

$$W_{00}^i = \{n_0 \in N_0 \mid \exists n \in W_0^{i-1}, (n_0, n) \in E\}$$

$$W_{01}^i = \{n_1 \in N_1 \mid (n_1, n) \in E \Rightarrow n \in W_0^{i-1}\}$$

The sequence W_0^i is non-decreasing. Since N is a finite set, for some i , $W_0^i = W_0^{i+1}$; let I be the least such i . W_0^I can be computed in $O(|E|)$ time. It is clear that $W_0^I \subseteq W_0$; before we prove that $W_0 = W_0^I$ we'll need a couple of definitions.

Definition 3.4.1 Given a graph-game $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), \phi \rangle$, and a subset of nodes $S \subseteq N$, if every node $n \in S$ has an edge in S , i.e., if there exists $n' \in S$ such that $(n, n') \in E$, then the **subgame** of \mathcal{G} on the node-set S is said to be well defined and is the graph game $\mathcal{G}_S = \langle G' = (N' = N'_0 \sqcup N'_1, E' = E'_0 \sqcup E'_1), \phi' \rangle$, where $N'_0 = S \cap N_0$, $N'_1 = S \cap N_1$, $E'_0 = E_0 \cap (N'_0 \times N'_1)$, $E'_1 = E_1 \cap (N'_1 \times N'_0)$, and ϕ' is a Boolean formula over S such that for $C \subseteq S$, $C \models \phi' \Leftrightarrow C \models \phi$. If ϕ is a Rabin or Streett formula, pair (L_i, U_i) of \mathcal{G} gets transformed to $(L'_i, U'_i) = (L_i \cap S, U_i \cap S)$ in \mathcal{G}_S .

Definition 3.4.2 Given a graph-game $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), \phi \rangle$ and a subgame $\mathcal{G}' = \langle G' = (N' = N'_0 \sqcup N'_1, E' = E'_0 \sqcup E'_1), \phi' \rangle$ of \mathcal{G} , we say **player- i controls the exits from \mathcal{G}' , or controls \mathcal{G}'** , if all edges out of subgame \mathcal{G}' are from player- i nodes, i.e., if $(n_1, n_2) \in E$, and $n_1 \in N'$ and $n_2 \in N \setminus N'$, then $(n_1, n_2) \in E_i$.

We can then observe the following easy lemma:

Lemma 3.4.1 Given a graph-game $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), \phi \rangle$ and a subgame $\mathcal{G}_S = \langle G' = (N' = N'_0 \sqcup N'_1, E' = E'_0 \sqcup E'_1), \phi' \rangle$ of \mathcal{G} on node-set $S \subseteq N$ controlled by player- i , $i \in \{0, 1\}$, if player- i wins in \mathcal{G}_S from $W'_0 \subseteq S$, then he also wins in \mathcal{G} from W'_0 by the same strategy, i.e., $W'_0 \subseteq W_0$.

We now return to proving that for the open game $W_0^I = W_0$.

Proposition 3.4.2 The set of nodes $N \setminus W_0^I$ induces a subgame of \mathcal{G} , and player-1 controls the exits from the subgame.

Proof: Let $N' = N \setminus W_0^I$. Consider $n \in N \setminus W_0^I$.

If n is a player-1 node and all its edges are to $N \setminus N'$ then $n \in W_{01}^{I+1}$. But $W_{01}^{I+1} \setminus W_0^I = \emptyset$, a contradiction. Thus every player-1 node in N' has an edge in N' .

If n is a player-0 node with an edge to $N \setminus N'$, again $n \in W_{00}^{I+1}$, but $W_{00}^{I+1} \setminus W_0^I = \emptyset$. Thus every player-0 node in N' has all edges from it to nodes in N' .

Therefore N' induces a subgame of \mathcal{G} controlled by player-1. ■

Since $F \subseteq W_0^I$, $N' = N \setminus W_0^I \subseteq \overline{F}$. By Prop. 3.4.2, player-1 controls the subgame on the node-set N' , and since she can stay out of F forever she wins from N' , i.e., $W_1 = N'$. Therefore, $W_0 \subseteq W_0^I = \emptyset$, and $W_0 = W_0^I$.

Both players have memory-less winning strategies: for a player-0 node $n_0 \in W_0$ the edge that caused its inclusion in W_{00}^i would suffice, and for a player-1 node $n_1 \in W_1$ an edge from n_1 that lies wholly within W_1 would suffice.

Since open and closed game are complementary, we have seen that we can decide open and closed games, and derive memory-less strategies for the winner in time linear in the size of the game graph. For the subsequent sections we shall assume the following two procedures:

1. $Open(G, F, i)$, for a game graph G and F a subset of its nodes, returns the nodes from which player- i can win assuming that player- i 's objective is to visit a node from F .
2. $Closed(G, T, i)$, for a game graph G and T a subset of its nodes, returns the nodes from which player- i can win by ensuring that the play never leaves the set T (T for “trap”).

3.5 Buchi and Co-Buchi games

In a Buchi graph game $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), F \rangle$, a play is a win for player-0 if it includes infinitely many visits to F . Let $W_0^0 = N$. For $i \geq 1$ we compute \widehat{W}_0^i , W_1^i , and W_0^i as follows.

$$\widehat{W}_0^i = Open(\mathcal{G}_{W_0^{i-1}}, F \cap W_0^{i-1}, 0)$$

$$T_1^i = W_0^{i-1} \setminus \widehat{W}_0^i$$

$$W_1^i = Open(\mathcal{G}_{W_0^{i-1}}, T_1^i, 1)$$

$$W_0^i = W_0^{i-1} \setminus W_1^i$$

\widehat{W}_0^i is the set of nodes in the subgame $\mathcal{G}_{W_0^{i-1}}$ from which player-0 can ensure a visit to F . If the play remains in the subgame $\mathcal{G}_{W_0^{i-1}}$, player-1 wins from the set T_1^i , since she can ensure that the play stays out of F (trap it in \overline{F}). And, here (in W_0^{i-1}), if player-1 can reach T_1^i she wins also, namely from the set W_1^i . The remaining nodes,

W_0^i , induce a subgame of $\mathcal{G}_{W_0^{i-1}}$ (and hence \mathcal{G}), exits out of which are controlled by player-0.

Player-1 clearly wins from T_1^1 , and hence from W_1^1 . Player-1 can win from W_1^2 within the subgame $\mathcal{G}_{W_0^1}$; if the play were to leave W_0^1 it would enter W_1^1 , from where player-1 wins. By induction on i , it can be shown that player-1 wins from $W_1 = \cup_{i \geq 1} W_1^i$.

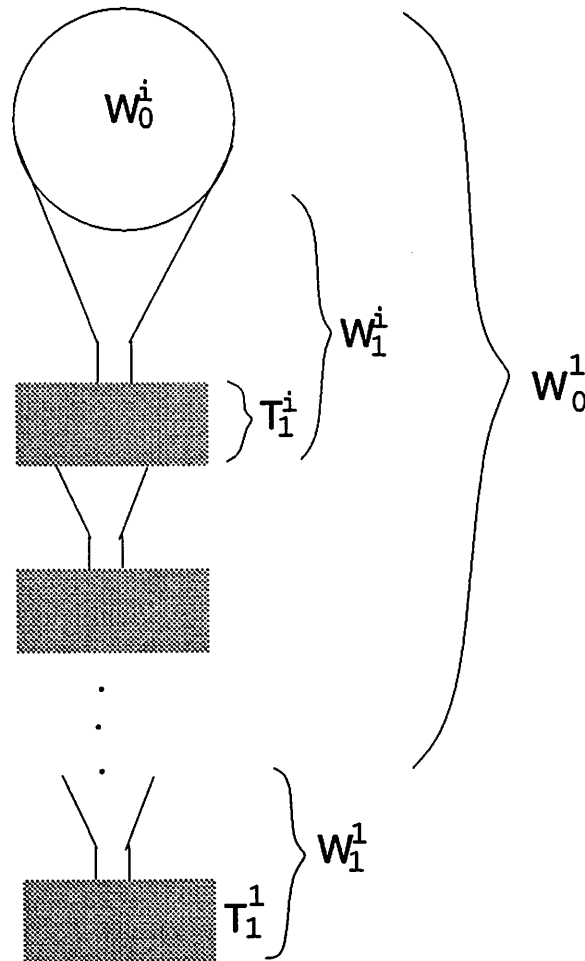


Figure 3.6: The partition of winning nodes in a Buchi game

Intuitively, $\cup_{i \geq 1} W_1^i$ is comprised of funnels ($W_1^i \setminus T_1^i$) and traps (T_1^i) as shown in Figure 3.6. If the play remains in a trap, player-1 wins by keeping the play out of F . Player-0 can however cause a descent into a funnel/trap system indexed by a

smaller number from where player-1 funnels the play into another trap. The play may descend this way only a finite number of times, as it will eventually reach T_1^1 from where no more descents are possible; player-1 wins from W_1 as eventually the play will remain forever in a trap.

Since W_0^i is a non-increasing sequence, for some i , $W_0^i = W_0^{i+1}$. Let I be the least such i . The claim is that player-0 wins from W_0^I . Since $W_0^{I+1} = W_0^I$, $W_1^I = \emptyset$, and $\widehat{W}_0^I = W_0^I$, and from every node in W_0^I player-0 can ensure a visit to F . Since W_0^I induces a subgame controlled by player-0, every player-0 node in $F \cap W_0^I$ has an edge into W_0^I , and all edges from player-1 nodes in $F \cap W_0^I$ are into W_0^I . Player-0 can thus repeatedly visit F from W_0^I , and hence wins the game from W_0^I .

Therefore player-0 wins from $W_0 = W_0^I$ and player-1 from $W_1 = \cup_{j=1}^{I-1} W_1^j$, and both have memory-less winning strategies. Since $1 \leq I \leq |N|$, deciding a Buchi or Co-Buchi game takes $O(|N||E|)$ time. It is interesting that given a partition of the nodes $N = W_0 \sqcup W_1$ in a Buchi game into nodes from which player-0 and player-1 win, we can derive a memory-less strategy for player-0 in linear ($O(|E|)$) time, whereas it is not clear how to avoid deciding the game again to derive a memory-less strategy for player-1 from the set W_1 . However, given the winning nodes for a game in $G_\delta \cap F_\sigma$, we can derive memory-less strategies for both players in linear time.

3.6 1-pair Rabin and Streett games

In a 1-pair Rabin game $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), L_1 \wedge \neg \overline{U}_1 \rangle$, a play is a win for player-0 if it includes infinitely many visits to L_1 and is eventually contained in U_1 . We compute the nodes in U_1 from which player-0 can ensure that every play remains in U_1 and visits L_1 infinitely; and, any nodes from which player-0 can reach such nodes is also winning for him. This is somewhat analogous to the case of computing the set of states in an ω -automaton from which the language generated is non-empty, where we compute the strongly connected components that contain an accepting strongly connected set, and then the states that can reach such SCCs.

We first compute $W_0^0 = \text{Close}(\mathcal{G}, U_1, 0)$, the set of nodes in U_1 that player-0 can trap the play in. W_0^0 induces a subgame of \mathcal{G} controlled by player-0, and he can win

in this subgame $\mathcal{G}_{W_0^0}$ provided he can ensure visits to L_1 infinitely often. Thus he wins from the winning nodes in the Buchi subgame $\mathcal{G}_{W_0^0}$ by a memory-less strategy. Let W_0^1 be the set of nodes in W_0^0 from which he can win the Buchi game $\mathcal{G}_{W_0^0}$. He can clearly also win from $W_0 = \text{Open}(\mathcal{G}, W_0^1, 0)$ with a memory-less strategy. $N \setminus W_0$ is a subgame, exits out of which are controlled by player-1 (and all exits are to winning nodes for player-0). Therefore, a winning strategy for either player in this subgame will also be a winning strategy in the game \mathcal{G} . We recursively repeat the computation of W_0^0 , W_0^1 , and W_0 , in the subgame $\mathcal{G}_{N \setminus W_0}$, and so on.

Finally, in some subgame, let it be $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), L_1 \wedge \neg \overline{U}_1 \rangle$ without loss of generality, $W_0 = \emptyset$.

We claim that player-1 wins here by a memory-less strategy. Within the subgame $\mathcal{G}_{W_0^0}$ she wins by her memory-less strategy to satisfy $\neg L_1$. If player-0 gets the play to stray out of W_0^0 into $N \setminus W_0^0$, she can force a visit to \overline{U}_1 . Thus either:

1. the play remains eventually in W_0^0 , and player-1 wins by satisfying $\neg L_1$, or
2. the play enters $N \setminus W_0^0$ infinitely often and player-1 wins again by visiting \overline{U}_1 infinitely often.

Thus, either way player-1 is able to win by satisfying her winning condition $\overline{U}_1 \vee \neg L_1$.

The above algorithm to decide the 1-pair Rabin game has time complexity $O(n^2|E|)$ (W_0^0 is computed in $O(|E|)$ time; W_0 in $O(n|E|)$ time, at most n times). It is interesting to note that given a partition of the nodes of a 1-pair Rabin game into the winning nodes for player-0 and player-1, we can derive a memory-less strategy for player-1 (as above) in $O(n|E|)$ time, whereas it is not clear how to derive player-0's strategy without repeating the computation to decide the game again.

Our algorithm to decide a 1-pair Rabin game is different from Emerson and Jutla's algorithm [14] to decide a 1-pair Rabin game, although they share the same complexity. However, our algorithm is simpler, easier to prove the correctness of, and is constructive in the sense that we also derive player-1's memory-less winning strategy. We present their algorithm next and supply our constructive correctness proof.

3.6.1 Emerson-Jutla's algorithm

Let $W_0^0 = N$. For $i \geq 1$, compute W_0^{ia} , W_0^{ib} , and W_0^i as follows:

$$W_0^{ia} = \text{Open}(\mathcal{G}_{W_0^{i-1}}, L_1 \cap W_0^{i-1}, 0)$$

$$W_1^{ib} = \text{Open}(\mathcal{G}_{W_0^{i-1}}, W_0^{i-1} \setminus W_0^{ia}, 1)$$

$$W_0^i = \text{Close}(\mathcal{G}_{(W_0^{ia} \setminus W_1^{ib})}, U_1 \cap (W_0^{ia} \setminus W_1^{ib}), 0)$$

W_0^{ia} is the set of nodes from which player-0 can ensure a visit to L_1 , and of these nodes, from those in W_0^{ib} player-1 can take the play out of W_0^{ia} . $W_0^{ia} \setminus W_1^{ib}$ induces a subgame controlled by player-0, and W_0^i is the set of nodes in this subgame that are in U_1 and from which player-0 can ensure that the play never leaves U_1 . W_0^i again induces a subgame controlled by player-0. For increasing i , W_0^i is a non-increasing sequence, and for some $i \geq 0$, $W_0^i = W_0^{i+1}$.

If I is the least such i and W_0^I is non-empty, then it is clear that player-0 has a memory-less winning strategy from W_0^I . Also, nodes from which player-0 can “reach” W_0^I , namely $W_0^{I'} = \text{Open}(\mathcal{G}, W_0^I, 0)$, are also winning for him. $N \setminus W_0^{I'}$ is a subgame exits out of which are controlled by player-1 (and all exits are to winning nodes for player-0), and we recursively repeat the above computation in the subgame. The form of Emerson-Jutla's algorithm is different from ours: they first enforce “touching” L_1 and then remaining in U_1 ; we do the reverse. Their proof of correctness is based on properties of temporal fixed-point logic operators and is not intuitive. We give below a new game-theoretic proof of the correctness of their algorithm.

In some final subgame, the set of winning nodes for player-0 will be empty. We argue that player-1 has a memory-less winning strategy in this subgame. Let the final subgame be $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), L_1 \wedge \overline{U_1} \rangle$.

Let $W_0^{i-1} \setminus W_0^i$ define the i^{th} layer; $W_0^{i-1} \setminus W_0^i = W_1^{ib} \sqcup W_1^i$, where $W_1^i = W_0^{i-1} \setminus W_0^i \setminus W_1^{ib}$. In W_1^{ib} player-1 has a memory-less strategy to ensure that the play is eventually contained in $\overline{L_1}$. Thus, if the play remains in W_0^{i-1} and enters W_1^{ib} , player-1 wins. If the play visits the i^{th} layer but not W_1^{ib} , i.e., it visits W_1^i , then player-1 can ensure that the play visits $\overline{U_1}$. Therefore, if the play remains in a certain layer player-1 ensures that she satisfies the condition $\overline{U_1} \vee \neg L_1$.

What changes of layers are possible? Recall that W_0^i is a subgame controlled by player-0. If the play leaves the i^{th} layer from W_1^{ib} it has to go to a layer indexed by a lower i ; call this a descent. Since exits from W_1^{1b} are controlled by player-1 she wins from there. Thus if the play always descends the play will eventually come to W_1^{1b} and player-1 will win by guaranteeing $\neg L_1$. During any ascent of layer, player-1 can ensure that the play transits through \overline{U}_1 . Therefore player-1 wins even if the play changes layers.

3.7 General Rabin and Streett games

In this section we consider deciding graph games where the winning condition is specified by two or more Rabin or Streett pairs. If we consider Streett games where the winning condition is specified with two or more pairs, it is no more true that the winner necessarily has a memory-less strategy. Consider the game graph of Figure 3.7 with player-0's winning condition specified by two Streett pairs: $(\{B\}, \emptyset)$ and $(\{C\}, \emptyset)$. The only node with a choice is the player-0 node labeled A . Fixing the edge from A results in a loss for him.

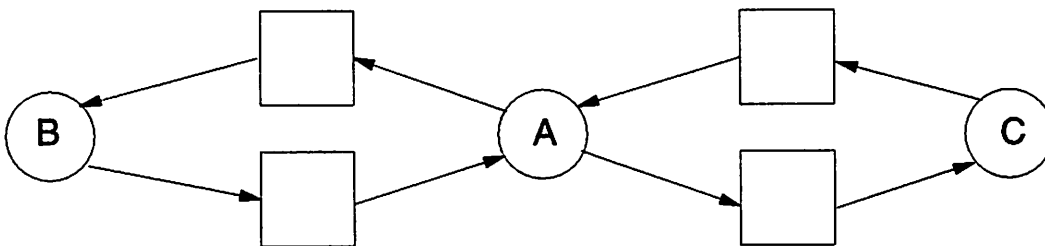


Figure 3.7: A Game where player-0 wins but only with memory

Unlike with deciding non-emptiness of Rabin ω -automata, for deciding a Rabin game it is not sufficient to decide the winner of appropriate 1-pair games—the interaction between the pairs has to be considered.

Consider the game graph of Figure 3.8, with winning condition for player-0 specified by the Rabin pairs: $(\{C\}, \{A, B, C, E\})$ and $(\{H\}, \{A, B, C, D, E, F, G, H\})$. From no node can player-0 ensure that every play satisfies one of the pairs (notice

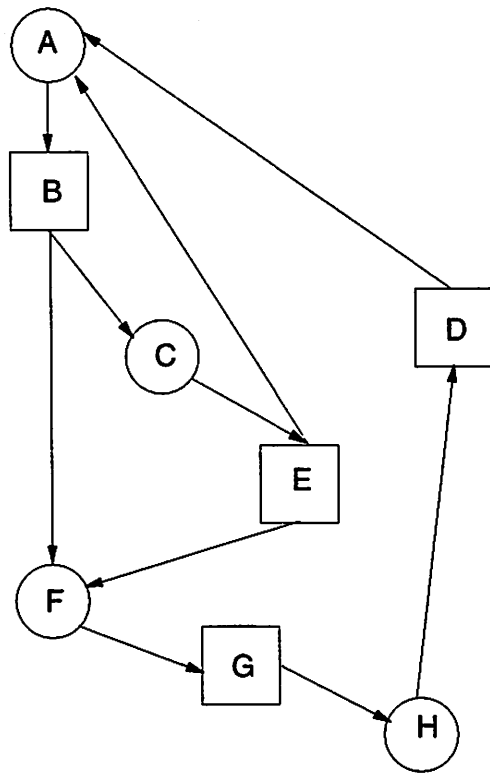


Figure 3.8: A Rabin game where 2 pairs *together* determine a win for player-0

player-0 has only one edge out of each node).

Therefore, the “interaction” between pairs has to be considered and the algorithm to decide a multiple pair Rabin game $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), \bigvee_{i=1}^h (L_i \wedge \neg \overline{U}_i) \rangle$ is recursive. For the j^{th} pair, (L_j, U_j) we compute the set of nodes in U_j , W_0^j , from which *player-0 can ensure that the play remains in U_j , and visits L_j infinitely often or satisfies one of the other pairs.*

Any node from which player-1 can take the play to a node in \overline{U}_j cannot be in W_0^j . W_0^j is computed as the limit of the non-increasing sequence, W_0^{ji} , where

$$W_0^{j0} = \text{Close}(\mathcal{G}, U_j, 0),$$

and for $i \geq 1$, W_0^{ji} is computed as follows:

$$A_0^{ji} = \text{Open}(\mathcal{G}_{W_0^{j(i-1)}}, L_j \cap W_0^{j(i-1)}, 0)$$

$$B_0^{ji} = \text{DecideRabinGame}(\mathcal{G}_{W_0^{j(i-1)} \setminus A_0^{ji}}, \{1, 2, \dots, h\} \setminus \{j\})$$

$$W_0^{ji} = \text{Close}(\mathcal{G}_{W_0^{j(i-1)}}, A_0^{ji} \cup B_0^{ji}, 0)$$

A_0^{ji} is the set of nodes in the subgame $\mathcal{G}_{W_0^{j(i-1)}}$ from where player-0 can ensure a visit to L_j . In the game $\mathcal{G}_{W_0^{j(i-1)}}$, in the subgame on the node set $W_0^{j(i-1)} \setminus A_0^{ji}$, B_0^{ji} is the set of nodes from which player-0 can win by satisfying one of the other pairs. *DecideRabinGame* is the recursive procedure being defined here, that returns the winning nodes for player-0, given the game as the first argument, and the indices of the relevant pairs as the second argument (for an empty set of pairs it returns the emptyset). W_0^{ji} is the subset of $A_0^{ji} \cup B_0^{ji}$ in the subgame $\mathcal{G}_{W_0^{j(i-1)}}$ that player-0 can trap the play in.

To win from W_0^{ji} player-0 has to be again be able to either ensure a visit to L_j or satisfy the other pairs. The sequence $W_0^{j0}, W_0^{j1}, \dots$ is non-increasing and for some least I_j , $W_0^{jI_j} = W_0^{j(I_j+1)}$. It is clear that player-0 has a memory-less strategy from nodes in $W_0^j = W_0^{jI}$ to ensure that the play remains in U_j , and either visits L_j infinitely often or satisfies one of the other pairs, and thus win from W_0^j .

Lemma 3.7.1 *In a graph game $\mathcal{G} = \langle G = (N, E), \phi \rangle$, if player- i has a memory-less winning strategy from $X_1 \subseteq N$ and a memory-less winning strategy from $X_2 \subseteq N$, then he has a memory-less winning strategy from $X_1 \cup X_2$.*

By Lemma 3.7.1 (easy to prove), player-0 also has a memory-less winning strategy from $\cup_{k=1}^h W_0^k$ and hence from any nodes from where he can reach $\cup_{k=1}^h W_0^k$, i.e., from $Open(\mathcal{G}, \cup_{k=1}^h W_0^k, 0)$. The remaining nodes are a subgame controlled by player-1, and we recompute, as above, W_0^j and so on.

In some final subgame, let it be $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), \bigvee_{j=1}^h (L_j \wedge \neg \overline{U}_j) \rangle$, for each pair, $j \in \{1, 2, \dots, h\}$, $W_0^j = \emptyset$. We show next that player-1 wins in this subgame.

For $i \geq 1$, define the i^{th} layer to be the set $W_0^{j(i-1)} \setminus W_0^{ji}$, and $D_0^{ji} = W_0^{j(i-1)} \setminus (A_0^{ji} \cup B_0^{ji})$. The i^{th} layer is also the set $C_0^{ji} = Open(\mathcal{G}_{W_0^{j(i-1)}}, D_0^{ji}, 1)$. If the play remains in $W_0^{j(i-1)}$, and enters the i^{th} layer, player-1 wins as she can drive the play to D_0^{ji} and win in D_0^{ji} by beating the other pairs and, of course, the j^{th} pair as $D_0^{ji} \subseteq \overline{L}_j$. However, $\mathcal{G}_{W_0^{j(i-1)}}$ is controlled by player-0 and he can divert the play out of $W_0^{j(i-1)}$. In this case, the play would have to descend to a lower layer—a layer indexed by a lower i . If the play eventually descends out of layer 1 (out of W_0^{j0}), player-1 can drive the play to \overline{U}_j .

For an h -pair Rabin game where $h > 1$, as in Figure 3.7 (complement the pairs), player-1 may need memory to win. We synthesize here a strategy for player-1 that uses $h!$ amount of memory. For each of the Rabin pairs (L_j, U_j) , player-1 has her strategy against the j^{th} pair, as discussed above, that ensures she can either discharge \overline{U}_j , or eventually trap the play in layer $i \geq 1$ and \overline{L}_j and beat the other pairs.

Player-1's strategy to win the game uses her strategy against the various pairs as follows. She first plays her strategy against the 1st pair. Either the play reaches \overline{U}_1 or she wins the game. When the play reaches \overline{U}_1 , if it does, she switches to her strategy against the 2nd pair. In this manner, while playing her strategy against the j^{th} pair, if the play reaches \overline{U}_j , she switches to her strategy against the $(j+1)^{th}$ pair; from \overline{U}_h , she switches to her strategy against the 1st pair. Thus she wins, as she guarantees her winning condition $\bigwedge_{j=1}^h (\overline{U}_j \vee \neg L_j)$, with h amount of memory and her strategies

against the h pairs. Her total memory requirement is $h!$, as the strategy for each pair involves a subgame with one less pair and the 1-pair game admits a memory-less strategy.

Time complexity of *DecideRabinGame*

The dominant step in the computation of W_0^{ji} in the subgame $\mathcal{G}_{W_0^{j(i-1)}}$, is the computation of B_0^{ji} that involves a recursive call to *DecideRabinGame* on a smaller game with one less pair. Since $I_j \leq n$, computation of W_0^j requires at most n recursive calls. Deciding a game, may require computing the W_0^j 's at most n times. Therefore it takes $\leq n^2 h * n^2 (h-1) * \dots * n^2 2 * n^2 |E|$ time, which is $n^{2h} h! |E|$, and $O((n^2 h)^h |E|)$ time, to decide the Rabin game $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), \bigvee_{i=1}^h (L_i \wedge \neg \bar{U}_i) \rangle$.

3.7.1 Emerson-Jutla's algorithm

Emerson-Jutla's algorithm [14] differs from ours presented above: the steps are reordered, and the proof is a little more complicated, but the computational complexity of the algorithms is the same as ours. They again compute W_0^j , but start with $W_0^{j0} = N$, and for $i \geq 1$, compute:

$$A_0^{ji} = \text{Open}(\mathcal{G}_{W_0^{j(i-1)}}, L_j \cap W_0^{j(i-1)}, 0)$$

$$B_0^{ji} = \text{EJDecideRabinGame}(\mathcal{G}_{W_0^{j(i-1)} \setminus A_0^{ji}}, \{1, 2, \dots, h\} \setminus \{j\})$$

$$C_1^{ji} = \text{Open}(\mathcal{G}_{W_0^{j(i-1)}}, W_0^{j(i-1)} \setminus (A_0^{ji} \cup B_0^{ji}), 1)$$

$$W_0^{ji} = \text{Close}(\mathcal{G}_{W_0^{j(i-1)}}, U_j \cap (A_0^{ji} \cup B_0^{ji}), 0)$$

Note that B_0^{ji} is different as it calls *EJDecideRabinGame*. C_1^{ji} is the set of nodes in the subgame $\mathcal{G}_{W_0^{j(i-1)}}$, from which player-1 can ensure that the play reaches the set $W_0^{j(i-1)} \setminus (A_0^{ji} \cup B_0^{ji})$; and, therefore wins in the subgame $\mathcal{G}_{W_0^{j(i-1)}}$ from the set C_1^{ji} . W_0^{ji} is the set of nodes in U_j , and either A_0^{ji} or B_0^{ji} , that player-0 can trap the play in.

To win from W_0^{ji} player-0 has to be again be able to either ensure a visit to L_j or satisfy the other pairs. The sequence $W_0^{j0}, W_0^{j1}, \dots$ is non-increasing and for some

least I_j , $W_0^{jI_j} = W_0^{j(I_j+1)}$. Again, player-0 has a memory-less strategy from nodes in $W_0^j = W_0^{jI}$ to ensure that the play remains in U_j , and either visits L_j infinitely often or satisfies one of the other pairs. Also, player-0 wins from any nodes from where he can reach $\cup_{k=1}^h W_0^k$, i.e., from $Open(\mathcal{G}, \cup_{k=1}^h W_0^k, 0)$. The remaining nodes are a subgame controlled by player-1, and we recompute, as above, W_0^j and so on.

In some final subgame, for each pair $j \in \{1, 2, \dots, h\}$, $W_0^j = \emptyset$, and we show next that player-1 wins in this subgame. Let the final subgame be $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), \bigvee_{j=1}^h (L_j \wedge \neg \overline{U}_j) \rangle$. As before we shall define player-1's strategy against each pair, and compose her winning strategy from these pairs giving a strategy that uses at most $h!$ memory.

Her strategy against a pair is expectedly defined a little differently. Let $D_1^{ji} = W_0^{j(i-1)} \setminus (A_0^{ji} \cup B_0^{ji})$. Within subgame $\mathcal{G}_{W_0^{j(i-1)}}$, player-1 wins from D_1^{ji} , and C_1^{ji} , but player-0 could cause the play to stray out of the set $W_0^{j(i-1)}$. However, player-1 clearly wins from C_1^{j1} as she clearly has the play trapped in D_1^{j1} .

Let $E_1^{ji} = W_0^{j(i-1)} \setminus W_0^{ji} \setminus C_1^{ji}$. From E_1^{ji} player-1 can force a visit to \overline{U}_j . For $i \geq 1$, define the i^{th} layer to be the set nodes $E_1^{ji} \sqcup C_1^{ji}$. Thus, for $j \in \{1, 2, \dots, h\}$, $N = \cup_{i=1}^{I_j} E_1^{ji} \sqcup C_1^{ji}$. For $i > 1$, player-0 may be able to divert the play out of C_1^{ji} , but the play would have to *descend* to a lower layer—a layer indexed by a smaller i . If the play ever visits a node in $\cup_{i=1}^{I_j} E_1^{ji}$, player-1 can discharge \overline{U}_j . If the play forever remains in $\cup_{i=1}^{I_j} C_1^{ji}$, player-1 wins by guaranteeing that the play eventually gets trapped in \overline{L}_j , and satisfies each of the other pairs.

3.7.2 Deciding Chain games

In a chain game, both players' winning conditions can be written as Rabin formulae, and therefore both players have memory-less winning strategies from the nodes that they win from. In this section we evaluate the complexity of deciding a chain game and deriving winning strategies for both players.

Let $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), \bigvee_{i=1}^k (\neg L_i \wedge U_i) \rangle$ be a Rabin chain game; the sets comprising the pairs form a chain:

$$L_1 \subseteq U_1 \subseteq L_2 \subseteq U_2 \subseteq \dots \subseteq L_k \subseteq U_k$$

The winning condition, $\bigvee_{i=1}^k (\neg L_i \wedge U_i)$ can be expressed equivalently in our favored syntax as $\bigvee_{i=1}^k (L'_i \wedge \neg \overline{U'_i})$, where $L'_i = U_i \setminus L_i$ and $U'_i = \overline{L_i} = N \setminus L_i$. For p, q such that $1 \leq p < q \leq k$, $L'_p \cap U'_q = \emptyset$.

Deciding the game applying the algorithm *DecideRabinGame*, consider the computation of W_0^j for $1 < j \leq k$. Recall that W_0^j is the set of nodes in U'_j from which player-0 can ensure that the play remains in U'_j and either reaches L'_j infinitely often or satisfies one of the other pairs. Since $U'_j \cap L'_p = \emptyset$, for $1 \leq p < j$, the computation of B_0^{ji} involves a recursion on the set of pairs with indices greater than j , i.e., the pairs indexed by the set $j < p \leq k$. Therefore the dominant step in deciding a chain game is the computation of $W_0^1 (B_0^{1i})$, and the chain game can be decided in time $n^2 * n^2 * \dots * n^2 |E|$, or $O(n^{2k} |E|)$ time.

It is also easier to derive the winning strategy for player-1 in a chain game. Her winning strategy is just her strategy against the Rabin pair (L'_1, U'_1) . Even upon reaching $\overline{U'_1}$, player-1 continues with her strategy against Rabin-pair (L'_1, U'_1) . There are two possibilities:

1. Either the play eventually remains in W_0^{10} and player-1 wins by trapping the play in $\overline{L'_1}$ and beating the other pairs, or
2. the play visits $\overline{U'_1}$, i.e., L_1 , infinitely often. This is again a win for her as her winning condition is:

$$\neg \bigvee_{j=1}^k (\neg L_j \wedge U_j) \equiv (L_1) \bigvee_{j=1}^{k-1} (\neg U_j \wedge L_{j+1}) \vee (\neg U_k)$$

Since player-1 sticks to her strategy against the first pair, it can be shown formally by induction on the number of pairs that her strategy is memory-less.

3.7.3 Deciding a Rabin game in “Rabin Index time”

One of the main results of Chapter 2 was the translation $DR(n, h) \rightarrow DRC(nh^k, k)$. This algorithm can also be used to translate a Rabin game $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), \bigvee_{i=1}^h (L_i \wedge \neg \overline{U_i}) \rangle$ into an equivalent Rabin chain game $\mathcal{G}' =$

$\langle G' = (N' = N'_0 \sqcup N'_1, E' = E'_0 \sqcup E'_1), \bigvee_{i=1}^k (\neg L'_i \wedge U'_i) \rangle$, where k is the Rabin Index of the game language. The resulting chain game can be decided in $O(m^{2k}|E'|)$ time, where $m = |N'|$, or $O((nh^k)^{2k}h^k|E|)$, i.e., $O(n^{2k}h^{2k^2+k}|E|)$ time. This time-complexity may not compare favorably with $O(n^{2h}h^h|E|)$ —the time to decide the Rabin game via *DecideRabinGame*—especially if the Rabin Index is high. Furthermore, the memory-less strategy derived for player-0 in \mathcal{G}' may be needlessly large when compared to the memory-less strategy that \mathcal{G} admits for him. We show next how to decide the Rabin game and derive a memory-less strategy for player-0 in time that compares favorably with deciding the Rabin game (via *DecideRabinGame*) no matter what the Rabin Index of the game language is.

We exploit the special structure of the chain game (automaton) returned by DRA2DRCA (Section 2.10.2, Chapter 2), to devise a procedure to decide the winner in a Rabin game $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), \bigvee_{i=1}^h (L_i \wedge \neg \bar{U}_i) \rangle$ in time $O(n^{2k}h^k|E|)$, and derive a memory-less winning strategy for player-0 from his winning nodes, and a strategy that uses at most h^k amount of memory for player-1 from her winning nodes. Note $O(n^{2k}h^k|E|)$ compares favorably with $O(n^{2h}h^h|E|)$ no matter what the RI $k \leq h$ is.

Theorem 3.7.2 *Let $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), \bigvee_{i=1}^h (L_i \wedge \neg \bar{U}_i) \rangle$ be a Rabin game with game language of Rabin Index k , and $\mathcal{G}' = \langle G' = (N' = N'_0 \sqcup N'_1, E' = E'_0 \sqcup E'_1), \bigvee_{i=1}^k (\neg C'_i \wedge D'_i) \rangle$ be the equivalent minimum-pair Rabin chain game produced by DRA2DRCA. If $W_0 \sqcup W_1 = N$, is the partition of the nodes N into respectively the nodes from which player-0 and player-1 win, then:*

1. *Player-0 has a memory-less winning strategy to win from W_0 ,*
2. *Player-1 has a winning strategy from W_1 that uses at most h^k memory, and is a memory-less strategy for her in the subgame of \mathcal{G}' on the node set $W_1 \times \{1, 2, \dots, h\}^k$, and*
3. *W_0, W_1 , and the players' winning strategies (as above) can be computed in time $O(n^{2k}h^k|E|)$ time, where $n = |N|$.*

Proof: By induction on the Rabin Index, k , of the game language.

Base case: the game language of \mathcal{G} has Rabin Index 1. The minimum pair equivalent Rabin chain game is $\mathcal{G}' = \langle G' = (N' = N'_0 \sqcup N'_1, E' = E'_0 \sqcup E'_1), \neg C'_1 \wedge D'_1 \rangle$, and the winning condition may be rewritten as $L'_1 \wedge \neg \overline{U'_1}$, where $L'_1 = D'_1 \setminus C'_1$ and $U'_1 = \overline{C'_1}$. Recall, \mathcal{G}' has h copies of the game graph G , i.e., $N' = N \times \{1, 2, \dots, h\}$, where edges from nodes in C'_1 are redirected to point from copy i to copy $i + 1$.

Consider the behavior of *DecideRabinGame* on \mathcal{G}' . The first step, computation of $W_0^{10} = \text{Close}(\mathcal{G}', \overline{C'_1}, 0)$, “decouples” the copies of \mathcal{G} into h disjoint subgames. Thus, $W_0^1 = \sqcup_{i=1}^h W_{0i}^1$, is comprised of h disjoint sets, each existing wholly within a copy of \mathcal{G} . Since W_{0i}^1 is essentially a subgame of \mathcal{G} controlled and won by player-0 with a memory-less strategy, he has a memory-less strategy to win from $\cup_{i=1}^h \Pi_N(W_{0i}^1)$, and $Z_0 = \text{Open}(\mathcal{G}, \cup_{i=1}^h \Pi_N(W_{0i}^1), 0)$ in \mathcal{G} , and $Z_0 \times \{1, 2, \dots, h\}$ in \mathcal{G}' .

Induce a subgame of \mathcal{G}' on the node set $N' \setminus (Z_0 \times \{1, 2, \dots, h\})$. On this subgame again compute W_0^{10} , W_0^1 , Z_0 , and so on. In some final subgame, $W_0^1 = \emptyset$, and we claim that player-1 wins from each of the nodes in this subgame (let it be \mathcal{G}' without loss of generality). In W_{0i}^{10} , player-1 has a memory-less strategy to win in the Buchi game $\mathcal{G}'_{W_{0i}^{10}}$. From $N' \setminus W_0^{10}$, she can force a visit to $\overline{U'_1}$, i.e., C'_1 . Therefore player-1 can guarantee her winning condition, $\overline{U'_1} \vee L'_1$ from all of N' and win.

Complexity analysis: W_0^{10} is computed in $O(h|E|)$ time, W_0^1 in $O(hn|E|)$ time, Z_0 in $O(|E|)$ time, and since a maximum of n subgames can be induced, the partition $N = W_0 \sqcup W_1$ and the associated winning strategies can be computed in $O(n^2h|E|)$ time.

Induction step: The basic idea is, as in *DecideRabinGame*, to compute W_0^j , the set of nodes in U'_j from which player-0 has a strategy to keep the play in U'_j and either visit L'_j infinitely often or satisfy the pairs indexed by $m > j$ (need only consider larger indices for a chain game). Before we describe the computation of W_0^j let us recollect some characteristics of DRA2DRCA. At the end of iteration i , $1 \leq i \leq k$, the game graph $\mathcal{G}_i = \langle G_i = (N_i, E_i), \phi_i \vee \psi_i \rangle$ has node set $N_i = N \times \{1, 2, \dots, h\}^i$, $\psi_i = \vee_{j=1}^i (\neg \Pi_{N_i}(C'_j) \wedge \Pi_{N_i}(D'_j))$, and ϕ_i is equivalent to the formula $\vee_{j=i+1}^k (\neg C'_j \wedge D'_j)$; pair (C'_i, D'_i) ((L'_i, D'_i)) gets defined on N_i . The game \mathcal{G}_i is equivalent to \mathcal{G} , and transitions between copies of \mathcal{G} in \mathcal{G}_i are from nodes in $\Pi_{N_i}(C'_i)$.

W_0^j is computed starting on \mathcal{G}_j . W_0^{j0} decouples the game \mathcal{G}_j into h^j disjoint subgames, one on each copy of \mathcal{G} in \mathcal{G}_j . W_0^j can therefore be computed separately on each of the copies by computing as in *DecideRabinGame*, A_0^{ji} , B_0^{ji} , and W_0^{ji} until $W_0^{ji} = W_0^{j(i+1)}$. The dominant step in each iteration, the computation of B_0^{ji} , by the induction hypothesis, can be done in $O(n^{2(k-j)}h^{k-j}|E|)$ time, as the Rabin Index of the subgame of $\mathcal{G}'_{W_0^{j(i-1)} \setminus A_0^{ji}}$ is $k-j$ (for $j < k$). Thus, computing W_0^j on each copy takes $O(n^{2(k-j)+1}h^{k-j}|E|)$ time, and on the h^j copies $O(n^{2(k-j)+1}h^k|E|)$ time. By Lemma 3.7.1, W_0^j being comprised essentially of memory-less strategies for player-0 on h^j disjoint subgames of \mathcal{G} , yields a memory-less strategy for him from $\Pi_N(W_0^j)$.

The computationally dominant among the W_0^j is W_0^1 taking $O(n^{2k-1}h^k|E|)$ time. From $W_0^1, W_0^2, \dots, W_0^k$, we can construct a memory-less winning strategy for player-0 in \mathcal{G} from $\cup_{j=1}^k \Pi_N(W_0^j)$, and $Z_0 = \text{Open}(\mathcal{G}, \cup_{j=1}^k \Pi_N(W_0^j), 0)$. Repeat the computation of W_0^j on the subgames $\mathcal{G}_{N \setminus Z_0}$, and $\mathcal{G}_{j_{N_j \setminus (Z_0 \times \{1, 2, \dots, h\}^j)}}$.

Note that the ‘‘copy replication’’ property holds of the subgames also. There are at most n such subgames, and W_0 can be determined in $O(n^{2k}h^k|E|)$ time. If $W_1 \neq \emptyset$, in some final subgame on the node set $W_1 = N \setminus W_0$, $W_0^j = \emptyset$ for each j , $1 \leq j \leq k$.

We claim that player-1’s winning strategy against the first Rabin pair (L'_1, U'_1) is a strategy that uses h^k memory and is a memory-less strategy for her from $W_1 \times \{1, 2, \dots, h\}^k$ in \mathcal{G}' . Consider the computation of W_0^1 as described above on $\mathcal{G}_1 = \langle \mathcal{G}_1 = (N_1, E_1), \phi_1 \vee \psi_1 \rangle$. From $N_1 \setminus W_0^{10}$ player-1 has a memory-less strategy to drive the play to $\Pi_{N_1}(\overline{U'_1})$, and hence from $(N_1 \setminus W_0^{10}) \times \{1, 2, \dots, h\}^{k-1}$ a memory-less strategy to drive the play to $\overline{U'_1}$. W_0^1 on copy m , $1 \leq m \leq h$, can be written as $W_{0m}^1 = C_{0m}^{11} \sqcup C_{0m}^1 1 \sqcup C_{0m}^{12} \sqcup \dots \sqcup C_{0m}^{1I_1}$, where $C_{0m}^{1i} = \text{Open}(\mathcal{G}_{1_{W_{0m}^{1(i-1)}}}, D_{0m}^{1i}, 1)$. By the induction hypothesis, within D_{0m}^{1i} player-1 has a strategy to beat the Rabin pairs indexed 2 through k , i.e., ensure $\neg\phi_1$, and this strategy uses h^{k-1} amount of memory, and is a memory-less strategy for her in the subgame of \mathcal{G}' on the node set $D_{0m}^{1i} \times \{1, 2, \dots, h\}^{k-1}$. From $(C_{0m}^{1i} \setminus D_{0m}^{1i}) \times \{1, 2, \dots, h\}^{k-1}$, she has a memory-less strategy to reach $D_{0m}^{1i} \times \{1, 2, \dots, h\}^{k-1}$.

Player-1’s strategy on each copy of \mathcal{G} in \mathcal{G}_1 thus requires h^{k-1} and is a memory-less strategy on an appropriate subgame of \mathcal{G}' ; the h copies of \mathcal{G} in \mathcal{G}_1 together yield a winning strategy for player-1 of the desired form. ■

The special structure of the game returned by DRA2DRCA can be exploited to not only decide a Rabin game faster via *DecideRabinGame*, but it also makes it particularly suited to a parallel implementation. All the computations are restricted to copies, and the computation of W_0^j on the copies in \mathcal{G}_j can happen in parallel; the information flow (arising from the recursion) also gives the algorithm a nice hierarchical property.

3.7.4 Putting it in perspective

As we saw, Rabin games admit memory-less winning strategies for player-0, and both players' winning strategies can be memory-less in chain games. Emerson [13] proved the existence—without synthesizing a strategy—of a memory-less strategy for player-0 in a Rabin game, and his proof can be inherited with almost no modification to yield the following theorem and corollary.

Theorem 3.7.3 *Let $\mathcal{G} = \langle G, \phi \rangle$ be a graph game that is negative union closed. Then player-0 has a memory-less strategy.*

Corollary 3.7.4 *(see also [34]) Let $\mathcal{G} = \langle G, \phi \rangle$ be a graph game that is both positive and negative union closed. Then player-0 and player-1 have memory-less winning strategies from their winning nodes.*

It follows from Theorem 3.7.3 [13] that deciding if player-0 wins a Rabin game is in *NP*, and deciding the winner in a Chain game is in $NP \cap \text{Co-NP}$. Emerson & Jutla [14] proved that the problem of deciding if player-0 (player-1) wins a Rabin game is NP-hard (Co-NP-hard).

Our main technical contribution is an algorithm to decide a Rabin game that is polynomial in the number of states and exponential only in the Rabin Index, of time complexity $O(n^{2k}h^k|E|)$, and is essentially optimum because the problem is NP-hard.

McNaughton, in a very nice paper [34], introduced game graphs and considered games with Muller winning condition. His paper is particularly noteworthy for its clean and clear development of game graphs, and useful notions such as subgames. He clarified and gave a simple exposition of the earlier work [7, 20] on deciding Muller

games, and the existence of Latest Visitation Record (LVR)-based—a particular form of memory-based—strategies for both players in Muller games.

Besides the technical merits of our synthesis algorithm, inspired by McNaughton, we believe we have given a simple, natural, and intuitive treatment of Rabin and Streett games. The form of our algorithm is related to and influenced by that of [14], where they consider the problem of checking the nonemptiness of tree automata. There is an intimate connection between tree automata and games, and this connection and the relevance of our results to tree automata is treated in the next section. In [14], the problem of deciding tree automata nonemptiness is viewed as a pseudo-model checking in a temporal logic with a fixed point construct. Their algorithm is exponential in the number of pairs and polynomial in the number of states, their proof technique is more complicated, and their algorithm is not constructive—they appeal to Martin’s theorem on the determinacy of Borel games to deduce the winner in a Streett game, and do not exhibit the strategy for the player with Streett winning condition.

[38] is another algorithm that is exponential in the number of pairs and based on translating the problem into one of checking nonemptiness of a finite tree automata. [51] considers the supervisory control problem of Rabin automata and adapts [14]’s algorithm, but supplies a different proof. [38, 51] also do not construct a strategy for the player with Streett winning condition.

One way to extract a winning strategy for the player with the Streett winning condition is to translate the game into an equivalent Chain game. McNaughton [34] provides an algorithm for deciding a chain game that is exponential in the number of nodes of the game graph. Recently, Puri [39] has devised an algorithm to decide a chain game that is exponential in the number of nodes, and is based on iteratively improving players’ strategies.

3.8 Games and tree automata

Tree automata—finite automata accepting infinite trees—were introduced by Rabin in 1969 and suggested as the tool to solve Church’s problem [40]. We consider

finite automata on labeled, infinite binary trees¹. The set $\{0, 1\}^*$ can be viewed as the infinite binary tree, where the root node is the empty string ϵ and each node η has two successors: the 0-successor $\eta 0$ and the 1-successor $\eta 1$ [14]. An infinite path through the tree is a sequence $\eta \in \{0, 1\}^\omega$. If Σ is a finite alphabet, a Σ -valued tree is a labeling $t : \{0, 1\}^* \rightarrow \Sigma$.

A finite automaton \mathcal{A} on infinite binary Σ -valued trees (henceforth tree automaton) is $\mathcal{A} = \langle T, \phi \rangle$, where T is a transition structure and ϕ is the acceptance condition. The transition structure is $T = \langle Q, q_0, \Sigma, \delta \rangle$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, Σ is a finite alphabet, and $\delta \subseteq Q \times \Sigma \times Q \times Q$ is the transition relation. If the transition relation is alternately expressible as a function $\delta : Q \times \Sigma \rightarrow Q \times Q$ the tree automaton is termed deterministic, and nondeterministic otherwise. The acceptance condition is a Boolean formula over the states Q (as with ω -automata), and using (for instance) a Rabin formula for the acceptance condition gives rise to a Rabin tree automata (RTA).

A run of \mathcal{A} on a Σ -valued tree t is a Q -valued tree $r_t : \{0, 1\}^* \rightarrow Q$, such that $r_t(\epsilon) = q_0$ and for all $\eta \in \{0, 1\}^*$, $(r_t(\eta), t(\eta), r_t(\eta 0), r_t(\eta 1)) \in \delta$. We say that \mathcal{A} *accepts* tree t provided in the run r_t of t on \mathcal{A} , for every infinite Q -labeled path α of r_t , $\phi[\text{inf}(\alpha)] = \text{true}$. The *language* of a tree automaton $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$, denoted $L_t(\mathcal{A})$ is the set of all Σ -valued trees that it accepts.

There is a close relation between tree automata and two-person Gale-Stewart games.

A tree-automaton can be viewed as a game between the two players: the Σ -player (player-0) and the path-player (player-1). More formally, given a deterministic TA $\mathcal{A} = \langle (Q, q_0, \Sigma, \delta), \phi \rangle$, define the corresponding game automaton as $\mathcal{G}' = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta'), \phi \rangle$ where $\Sigma_0 = \Sigma$, $\Sigma_1 = \{0, 1\}$, and $\delta(q, \sigma) = (\delta'(q, (\sigma, 0)), \delta'(q, (\sigma, 1)))$. Similarly, given a deterministic game automaton a corresponding DTA can be inferred. Hence, for a DTA the corresponding Game automaton (GA) is well-defined and unique and vice-versa.

A Σ -valued tree accepted by a tree automaton can be viewed as a strategy for

¹extension to k -ary trees is straightforward

the Σ -player in the corresponding game automaton. Hence, Player-0 has a winning strategy in the game automaton iff the corresponding tree automaton has a non-empty language. A regular strategy for player-0 defines a *regular* tree accepted by the tree automaton.

Can nondeterministic TA be translated to games? Let $\pi : \Sigma' \rightarrow \Sigma$ be a function between two finite alphabets Σ' and Σ . Given a Σ' -valued tree $t_1 : \{0, 1\}^* \rightarrow \Sigma'$, let $\pi(t_1)$ denote the Σ -valued tree $t_2 : \{0, 1\}^* \rightarrow \Sigma$ obtained by composing t_1 and π .

Lemma 3.8.1 *Suppose $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$ is a nondeterministic TA. Then there exists a DTA $\mathcal{A}' = \langle T' = (Q, q_0, \Sigma', \delta'), \phi \rangle$, and a projection $\pi : \Sigma' \rightarrow \Sigma$ such that $L(\mathcal{A}) = \{t | t' \in L_t(\mathcal{A}') \text{ and } t = \pi(t')\}$*

Proof: The construction is simple and a well known folk fact. $\Sigma' = \Sigma \times Q \times Q$, and for $a \in \Sigma$, $\pi(a, q_1, q_2) = a$. $\delta'(q, (a, q_1, q_2)) = (q_1, q_2)$ provided $(q_1, q_2) \in \delta(q, a)$. \mathcal{A}' can be completed if necessary by the addition of a “dead” state. ■

From Lemma 3.8.1 and the correspondence between DTA and games, we deduce that we can decide the non-emptiness of Rabin and Streett TA with our algorithms for deciding games, and extract a regular tree accepted by the TA if it has a non-empty language. For a DRTA $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \bigvee_{i=1}^h (L_i \wedge \neg \overline{U}_i) \rangle$, the translation implies that we can decide if $L_t(\mathcal{A}) \neq \emptyset$ in $O(n^{2k} h^k |\delta|)$, which is $O(n^{2k+1} h^k)$ time as $|\delta| = O(n)$ assuming Σ is of constant size, and where $n = |Q|$, and k is the Rabin Index of the game language of the game automaton corresponding to the DRTA \mathcal{A} .

However for nondeterministic TA the translation to a game automaton results in an alphabet of non-constant size, which in turn results in an equivalent graph game where the number of nodes is quadratic in the number of states of the TA, and the number of edges is also possibly larger than the size of the transition relation of the TA. We can avoid some of this blow-up by translating the TA $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$ directly into a graph game $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), \phi' \rangle$, where:

- $N_0 = Q, N_1 = \{(q, \sigma, q_1, q_2) | (q, \sigma, q_1, q_2) \in \delta\}$
- $E_0 = \{(q, (q, \sigma, q_1, q_2)) | (q, \sigma, q_1, q_2) \in \delta\}$

- $E_1 = \{((q, \sigma, q_1, q_2), q_1) \mid (q, \sigma, q_1, q_2) \in \delta\} \cup \{((q, \sigma, q_1, q_2), q_2) \mid (q, \sigma, q_1, q_2) \in \delta\}$
- ϕ' is defined to be such that for $S \subseteq N$, $S \models \phi' \Leftrightarrow S \cap N_0 \models \phi$. If ϕ is a Rabin or Streett formula, then a pair (L_i, U_i) in ϕ gets translated to pair $(L'_i, U'_i) = (L_i \sqcup (L_i \times \{(q, \sigma, q_1, q_2) \mid (q, \sigma, q_1, q_2) \in N_1 \text{ and } q \in L_i\}), U_i \sqcup (U_i \times \{(q, \sigma, q_1, q_2) \mid (q, \sigma, q_1, q_2) \in N_1 \text{ and } q \in U_i\}))$

Therefore $|E| = O(|\delta|)$, and $|N| = O(|\delta|)$ and we can check for the nonemptiness of an h -pair RTA in $O(m^{2k+1}h^k)$ time, where $m = |\delta|$ and k is the Rabin Index of the corresponding game language (of the graph game).

Our translations between different types of deterministic ω -automata can thus be used to also translate between different kinds of tree automata. Also, we can check for nonemptiness of a DRTA in time that is exponential in the Rabin Index of the game language of the corresponding game automaton. For a given DRTA, let us define the Rabin Index of the language of the tree automaton as the minimum number of pairs required to realize the language of the tree automaton as a DRTA. In fact, tree automata are more intricate than ω -automata, and the nondeterministic variants of all acceptance conditions are strictly more expressive than the deterministic variants; one can therefore also define the Rabin Index for nondeterministic TA.

While the general problem of the minimization of the number of pairs in a deterministic Rabin TA and nondeterministic RTA is more difficult than for deterministic ω -automata, we identify a special case that we can solve.

Definition 3.8.1 *We say a TA $\mathcal{A} = \langle T = (Q, q_0, \Sigma, \delta), \phi \rangle$ is trim provided for every $s \in Q$ the TA $\mathcal{A}_s = \langle T = (Q, s, \Sigma, \delta), \phi \rangle$ accepts some tree, i.e., $L_t(\mathcal{A}_s) \neq \emptyset$.*

Theorem 3.8.2 *Given a trim DRTA \mathcal{A} , let \mathcal{A}_G be the corresponding game automaton. Suppose \mathcal{B}_G is obtained from \mathcal{A}_G by minimizing the number of pairs. Then the DRTA \mathcal{B} corresponding to \mathcal{B}_G has the minimum number of pairs among DRTA accepting $L_t(\mathcal{A})$.*

Proof: Let \mathcal{M} be a minimum pair equivalent DRTA, i.e., such that $L_t(\mathcal{M}) = L_t(\mathcal{A})$. \mathcal{M} can be trimmed, i.e., states from which the language accepted is empty deleted, giving an equivalent DRTA with the same number of pairs.

We claim that $L(\mathcal{B}_G) = L(\mathcal{M}_G)$. Observe that in a trim TA every sequential accepting run is “part” of some accepting tree run. In a DTA a sequential run denotes a unique sequence in $(\Sigma \times \{0,1\})^\omega$. Since the language of the game automaton is exactly the set of accepting sequential runs it follows that $L(\mathcal{B}_G) = L(\mathcal{M}_G)$.

Therefore \mathcal{B} is a DRTA with no more pairs than the minimum pair DRTA \mathcal{M} , and is a minimum pair DRTA for \mathcal{A} . ■

3.9 Lower bounds for memory and strategy-size

While the player with the Rabin winning condition has a memory-less strategy, the player with the Streett or Muller condition may need memory to win. For a Streett game, we presented an algorithm that synthesized a strategy that uses h^s amount of memory, where h is the number of Streett pairs used to specify the winning condition, and s is the Streett Index of the game language. For Muller games on n nodes, $n!$ is sufficient amount of memory to implement a winning strategy for either player (see [34] for this synthesis procedure; alternately the translation from a DMA to DCA presented in Chapter 2 in conjunction with the algorithm to decide a Chain game discussed in this chapter yields a synthesis algorithm for Muller games).

How do these synthesis algorithms for Streett and Muller games perform? Is there an algorithm that can consistently synthesize strategies using less memory, i.e., a better upper bound? In this section, we consider lower bounds for the memory required to implement the winner’s strategy when his winning condition is a Streett or Muller formula. We shall aim to prove, when possible, lower bounds on the size of the smallest FSM implementing the winner’s strategy in a game on an ω -automaton. This also implies a lower bound on the memory required of a memory-based strategy— if m is a lower bound on the strategy size, then $\frac{m}{n}$ is a lower bound on the amount of memory required of a memory-based strategy. Considering only memory-based strategies overlooks the possibility of minimizing the strategy viewed as an FSM.

Memory Bounds for Games in G_δ

Consider a game defined by the following transition structure (GT) : $GT(n)$ has n states respectively labeled $1, \dots, n$. Player-0's alphabet is $\Sigma_0 = \{S | S \subseteq \{1, \dots, n\} \text{ and } |S| = \lceil \frac{n}{2} \rceil\}$, and Player-1's alphabet is $\Sigma_1 = \{1, \dots, n\}$. If Player-0 plays $S \in \Sigma_0$, then player-1 can play any $l \in S$ (if player-1 plays any $l \notin S$ she immediately forfeits the game), and l is the next state. The transition structure GT along with an acceptance condition completely defines a game. We will use the same transition structure, GT, more than once in this section with different acceptance conditions.

The n -state G_δ -game is defined by $GT(n)$ and an acceptance condition in either Streett or Muller form. In Streett form the pairs are $\{(S, \emptyset) | S \subseteq \{1, \dots, n\} \text{ and } |S| = \lceil \frac{n}{2} \rceil\}$. In other words a set $\psi \subseteq \{1, \dots, n\}$ is accepted provided $|\psi| \geq \lfloor \frac{n}{2} \rfloor + 1$. There are $\binom{n}{\lceil \frac{n}{2} \rceil}$ Streett pairs. In Muller form, the acceptance sets are all subsets of $\{1, 2, \dots, n\}$ with $\geq \lfloor \frac{n}{2} \rfloor$ elements, i.e., the upper half of the subset lattice.

In a game, in general the next state in the game automaton depends on: the current state, the choice of player-0 (from Σ_0), and the choice of player-1 (from Σ_1). It is noteworthy that in the transition structure GT, the next state is a function of the choice made by player-0 and player-1, and is independent of the current state from which this "1-step" play is made. For example, in $GT(4)$ if player-0 plays $\{2, 4\}$, player-1 may play either 2 or 4, and if player-1 plays (say) 2, the next state is 2.

Proposition 3.9.1 *Player-0 wins the n -state G_δ game using $\binom{n}{\lfloor \frac{n}{2} \rfloor - 1}$ memory.*

Proof: Player-0's memory at any instant of the game will be a subset of $\{1, 2, \dots, n\}$. Initially let the play be at state m_0 and the memory be $M_0 = \emptyset$. Player-0 picks $\sigma_0^0 \in \Sigma_0$ such that $m_0 \notin \sigma_0^0$. Let player-1 play to $m_1 \in \sigma_0^0$; clearly $m_1 \neq m_0$. M_1 , the current memory is set to $\{m_0\}$. Now player-0 picks $\sigma_0^1 \in \Sigma_0$ such that $\sigma_0^1 \cap (M_1 \cup \{m_1\}) = \emptyset$. Let player-1 play to $m_2 \in \sigma_0^1$; $m_2 \notin \{m_0, m_1\}$. M_2 is set to $M_1 \cup \{m_1\}$. In this manner, from the current state m_i , for $i < \lfloor \frac{n}{2} \rfloor$, player-0 plays σ_0^{i+1} such that $\sigma_0^{i+1} \cap (M_i \cup \{m_i\}) = \emptyset$. $M_{i+1} = M_i \cup \{m_i\}$ until $i = \lfloor \frac{n}{2} \rfloor - 1$. At this point, we reset the index i , i.e., let $m_0 = m_{\lfloor \frac{n}{2} \rfloor}$ and $M_0 = \emptyset$.

The strategy is winning for player-0, since from m_0 player-0 is able to ensure visits to $\lfloor \frac{n}{2} \rfloor$ additional distinct states, $m_1, \dots, m_{\lfloor \frac{n}{2} \rfloor}$. ■

Next, we establish a lower bound on the size of the strategy required in the G_δ -game.

Theorem 3.9.2 *Given $n \geq 2$, every winning strategy FSM for player-0 in the n -state G_δ -game has at least $\binom{n}{\lceil \frac{n}{2} \rceil}$ states.*

Proof: Suppose a winning strategy for player-0 has less than $\binom{n}{\lceil \frac{n}{2} \rceil}$ states. Since a strategy for player-0 is a Moore FSM, there exists $S \in \Sigma_0$ that is never selected by player-0. Player-1's strategy then is to always pick her choice from $\{1, \dots, n\} \setminus S$. She is guaranteed a "hit", as the only choice by player-0 that will deny her a hit, namely S , has been disallowed by assumption. She thus wins as the set of states visited infinitely often would be smaller than $\lceil \frac{n}{2} \rceil$.

Therefore player-0 has to choose each distinct $\sigma_0 \in \Sigma_0$; hence every winning strategy FSM for him (being Moore) has at least as many states as distinct outputs, i.e., at least $\binom{n}{\lceil \frac{n}{2} \rceil}$ states, which is also greater than $2^{\lfloor \frac{n}{2} \rfloor}$. ■

The upper bound for the winning strategy for player-0 in the n -node G_δ -game is $n \cdot \binom{n}{\lfloor \frac{n}{2} - 1 \rfloor}$ and the lower bound is $\binom{n}{\lceil \frac{n}{2} \rceil}$, asymptotically tight. Since the G_δ -game can be cast with both Muller and Streett conditions, we have Streett and Muller games in G_δ such that the smallest strategy is exponential in the number of states in the game automaton. The memory furnished by translating the DMA or DSA into an equivalent DBA may in fact be required.

Bounds for General Muller Games

In a general Muller game (i.e., not necessarily in G_δ), a sufficient amount of memory to implement a winning strategy is $n!$ [20, 34], which is $2^{n \log n}$, where n is the number of states in the game automaton. What is the lower bound?

Consider the following game (called the n -state general Muller (GM) game): the transition structure is $GT(n)$, and the acceptance condition is induced by the following assignment of $+$ and $-$ to the sets in the subset lattice: $S \subseteq \{1, 2, \dots, n\}$ is assigned $+$ if and only if $|S|$ is even; otherwise it is assigned $-$.

Proposition 3.9.3 *Player-0 wins the n -state GM game if n is even, and player-1 wins if n is odd.*

The strategy used is an LVR-based [20, 34] strategy, where LVR stands for least visitation record. The LVR is an n length vector of states of the game automaton, where the state indexed by 1 is the most recently visited state and the state indexed by n is the least recently visited state. Thus $R[1]$ is the current state. Assume that the next state is $m = R[j]$. The LVR at state m is $R' = R[n] \dots R[j + 1]R[j - 1] \dots R[1]R[j]$.

When n is even, player-0 plays the following strategy, called the *even-index strategy*: play $\sigma_0 \in \Sigma_0$ such that σ_0 is the set of labels of the even indexed states in the LVR. So if the LVR is R , player-0 plays $\{R[i] \mid i \text{ is even}\}$. To show that this yields a winning strategy for player-0 it suffices to show that the set of states visited infinitely often has even cardinality. Since player-0 plays the same set of indices of the LVR every time, no matter what player-1 does, there is going to be some largest index that she visits infinitely often; let this index be m ; m is even and $2 \leq m \leq n$. The set of states visited infinitely often, $\{R[i] \mid 1 \leq i \leq m\}$, would then be of size m —an even number.

Let us illustrate the even-index strategy on the 4-state GM game. Let us say the game is initially at state 1. Assume without loss of generality that the LVR is [4321]; the only check we need to make now is that 1 is the most recently visited state—the current state. At this point player-0 plays the set $\{2, 4\}$. Say player-1 now moves to state 2. The LVR becomes [4312]. Then player-0 plays $\{4, 1\}$. Say player-1 plays to state 4. The LVR becomes [3124]. Now player-0 chooses $\{2, 3\}$, and so on.

When n is odd, player-1 plays the following strategy: play $\sigma_1 \in \Sigma_1$ such that Σ_1 is the set of odd indexed states of the LVR. So if the LVR is R , then player-1 plays from the set $\{R[i] \mid i \text{ is odd}\}$, choosing the highest indexed element in case of a choice of more than one odd-indexed state. Note that since n is odd and player-0 has to choose a set of size $\lceil \frac{n}{2} \rceil$, player-0 has to present at least one odd-indexed state, and therefore player-1 is guaranteed a “hit” every time.

Since player-1 has a set of $\lceil \frac{n}{2} \rceil$ states to choose from, she is guaranteed a “hit” no matter what player-0 plays. It is easy to see that no matter what player-0 plays, player-1 can guarantee that the set of states visited infinitely often is of odd cardinality.

3.9.1 A lower bound for the General Muller game

In this section we attempt to establish an $n!$ lower bound for the size of a winning strategy for player-0 in the n -state GM game, for even n .

Definition 3.9.1 *In the GM-game on n states, we say player-0 presents the i^{th} index, if his play $S \subseteq \{1, 2, \dots, n\}$, where $|S| = \lceil \frac{n}{2} \rceil$, contains $R[i]$, i.e., $R[i] \in S$.*

Let us consider the GM-game for even n . Consider the following specific strategy for player-1. She plays the highest odd-indexed state, and if no odd-indexed states are presented to her, she picks the least even-indexed state (i.e., the state indexed by 2).

Against this strategy of player-1 we first claim that player-0 cannot present any odd index greater than 1 infinitely often. Suppose not for contradiction. Let the largest odd index presented infinitely often be k . It follows that the set of states visited infinitely often is of size k —an odd number—and therefore player-1 wins.

It is clear that player-0 should never present the index 1, as player-1 can play the state indexed in the LVR by 1 and the LVR will remain unaltered; this state can be eliminated from the strategy (there cannot be a self-loop under the state that presents index 1, as this would mean a loss for him) to result in a smaller strategy.

It thus follows that player-0 can only present the even indices infinitely often. Therefore, from some point onwards player-0 has to play $\{R[i] \mid i \text{ is even}\}$. This is the unique winning strategy for player-0.

Therefore to compute the lower bound, we need to see how many states the smallest strategy FSM employing the even-index strategy has.

We first claim that employing the even-index strategy FSM every distinct LVR can be reached, i.e., $n!$ different LVRs can be reached. Assume the current LVR is $R[n \dots 1]$. $R[2]$ and $R[1]$ can be exchanged if, of the even-indices presented, player-1 chooses 2. To exchange any two adjacent elements in the LVR, player-1 chooses the index n , until the two elements occupy indices 1 and 2, and then they can then be easily switched. After this, player-1 chooses the n^{th} index till the states (except the two exchanged states) have rotated back to their old indices. It thus takes $n + 1$ 1-step plays to exchange the two adjacent states. It follows that player-1 can force

any arbitrary permutation as the LVR, since we have shown that any transposition can be achieved.

Is the even-index strategy FSM for player-0 where each distinct LVR is a different state minimum-state? States with different output cannot clearly be equivalent. Consider two states with the same output. Can they be equivalent? If they are different states then they have different LVRs. The claim is that these states are not equivalent.

Let the least index where the two states differ, be i . We consider two cases:

1. i is even. Consider the choice of index i by player-1. Their outputs in the next state will be different.
2. i is odd. Consider the choice of index $i + 1$ by player-1. The claim is that the output in the next state will be different.

The two different strategies for player-1, one to force player-0 to play the even-index strategy, and the second to force $n!$ LVRs when player-0 plays the even-index strategy, seem to be incompatible. We think they can be “reconciled” into one strategy to give an $n!$ lower bound. We are currently working on the details, and are hence obliged to call the following two results conjectures.

Conjecture 3.9.4 *For every even $n \geq 2$, every winning strategy for player-0 in the n -state GM game has at least $n!$ states.*

Conjecture 3.9.5 *For every odd $n \geq 3$, every winning strategy for player-1 in the n -state GM game has at least $\frac{n!}{2}$ states.*

Recently it has been shown [12] that there are n -node Muller game graphs where a player needs $2^{O(n \log n)}$ memory to win, strengthening our conjecture.

3.9.2 Lower bound for general Streett games

The G_δ game already shows an exponential (in the number of states of the game automaton) lower bound on the strategy size for Streett games in G_δ . However there

is always a strategy FSM that is linear in the size of the game automaton, namely of size nh , where n is the number of states and h is the number of pairs in the game automaton. For general Streett games, our synthesis algorithm establishes an upper bound of $n2^{s \log h}$ on the strategy size, where s is the Streett Index of the game language. The question arises if a strategy exponential in the size of the game automaton is ever required.

Theorem 3.9.6 *For every $n \geq 1$ there exists a Streett graph game on $5n$ nodes and $2n$ pairs such that player-0 needs at least 2^n amount of memory to implement a winning strategy.*

Proof: For each $n \geq 1$, consider the graph game $\mathcal{G}_n = \langle G_n = (N_n = N_n^0 \sqcup N_n^1, E_n = E_n^0 \sqcup E_n^1), \phi_n \rangle$, where $N_n^0 = \{(i, x) | 1 \leq i \leq n, x \in \{A, B\}\}$, $N_n^1 = \{(i, x) | 1 \leq i \leq n, x \in \{0, a, b\}\}$,

$$\begin{aligned} E_n^0 &= \{((i, A), (i, a)) | 1 \leq i \leq n\} \\ &\sqcup \{((i, A), (i, b)) | 1 \leq i \leq n\} \\ &\sqcup \{((i, B), (i, 0)) | 1 \leq i \leq n\}, \end{aligned}$$

$$\begin{aligned} E_n^1 &= \{((i, c), (j, C)) | 1 \leq i \leq n \ \& \ j = i \bmod n + 1 \ \& \ c \in \{a, b\} \ \& \ C \in \{A, B\}\} \\ &\sqcup \{((i, 0), (j, C)) | 1 \leq i \leq n \ \& \ j = i \bmod n + 1 \ \& \ C \in \{A, B\}\}. \end{aligned}$$

The acceptance condition, ϕ_n , consists of $2n$ Streett pairs, two for each level $i \in \{1, 2, \dots, n\}$: $(\{(i, a)\} \Rightarrow \{(i, b)\})$, and $(\{(i, b)\} \Rightarrow \{(i, a)\})$.

The game graph G_n is shown in Figure 3.9. The game is won by player-0, if for each $i \in \{1, 2, \dots, n\}$, either both (i, a) and (i, b) are visited infinitely often, or neither is.

There is an interesting relationship between the family of games \mathcal{G}_n and the language-family used to prove the lower bound for the translation of DSA into DCA (Theorem 2.13.2). Associate with the edges in E_n the letter corresponding to the label of the terminal node; thus player-0's alphabet is $\Sigma_0 = \{0, a, b\}$, and player-1's alphabet is $\Sigma_1 = \{A, B\}$. In order to define a Gale-Stewart game, assume that playing from $\{a, b\}$ from a player-0 node labeled B , and playing 0 from a player-0 node

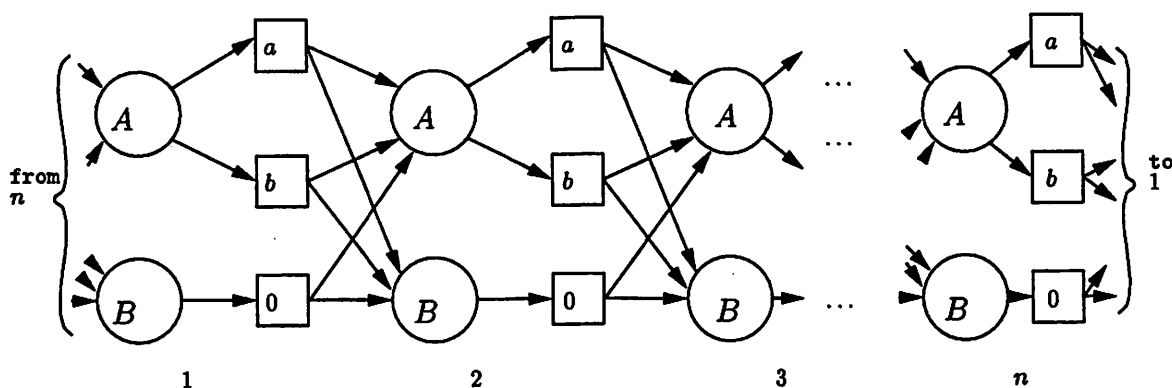


Figure 3.9: Streett game where player-0 needs 2^n memory

labeled A takes the play to a “dead” subgame from where player-1 wins trivially. Now project the game language, Γ_n , down to Σ_0 , i.e., from the winning plays for player-0, drop player-1’s choices—to get $\Gamma_{n\Sigma_0}$; each sequence so obtained, $\sigma \in \Gamma_{n\Sigma_0}$, is also in L_n of Theorem 2.13.2, and also every $\sigma \in L_n$ is included in $\Gamma_{n\Sigma_0}$ except for the first letter in the sequence.

It is easy to see that player-0 can win \mathcal{G}_n with 2^n amount of memory. For each of the player-0 nodes (i, A) he remembers what he played the last time the play was at (i, A) — a or b —and plays the other letter next. We show next, by induction on n , that he needs at least 2^n amount of memory to win in \mathcal{G}_n .

Base case: Recall that in a memory-based strategy for player-0, each node in his strategy graph is labeled by (the name of) some player-0 node in the game graph and the amount of memory used is determined by the number of occurrences of the most popular node from the game graph. It is clear that in \mathcal{G}_1 no memory-less strategy for player-0 earns him a win; therefore any memory-based strategy graph for him contains at least 2 nodes labeled A .

In the game graph \mathcal{G}_n , it is clear that player-1 can take the play to any player-0 node she wants, and if player-0 wants to win, to any player-1 node as well. Recall that in a strategy graph for player- i , $i \in \{0, 1\}$, the play finally settles in a SCC, and player- $1 - i$ can take the play to any node of the SCC. We shall prove by induction on n that every terminal strongly connected component (tSCC) of the winning (memory-based) strategy graph for player-0 in \mathcal{G}_n contains at least 2^n nodes labeled $(1, A)$, i.e.,

at least 2^n amount of memory to win. Note that the base case has been validated.

Induction step: Consider the subgame of \mathcal{G}_n on the node set $N_n \setminus \{(n, A), (n, a), (n, b)\}$. Let \mathcal{G}'_n denote this subgame. It is easy to see that \mathcal{G}'_n and \mathcal{G}_{n-1} are equivalent in the following sense:

1. Player-0 wins both games, and
2. A memory-based winning strategy for player-0 in \mathcal{G}_{n-1} can be easily translated to a memory-based winning strategy (MBWS) for player-0 in \mathcal{G}'_n using the same amount of memory, and vice-versa.

Now, in \mathcal{G}_n , player-1 can force the play to be in the subgame \mathcal{G}'_n , and by the induction hypothesis each tSCC of player-0's memory-based winning strategy graph will have at least 2^{n-1} nodes labeled $(1, A)$. Notice that it is also true in \mathcal{G}'_n that if player-0 plays to win player-1 can take the play to any node in \mathcal{G}'_n . Therefore each tSCC of player-0's MBWS graph contains all the node labels in \mathcal{G}'_n . Thus from each node in the tSCC C_1 of the MBWS of player-0 for \mathcal{G}'_n , player-1 can take the play to some designated player-1 node with label $(n-1, a)$, from where she can play the edge (say) $((n-1, a), (n, A))$ taking the play out of the subgame \mathcal{G}'_n .

(Say) player-0 now plays the edge $((n, A), (n, a))$, and the play reaches $(1, A)$. Player-1 can again restrict the play to subgame \mathcal{G}'_n , and again by the induction hypothesis the tSCC C_2 that the play settles to contains at least 2^{n-1} nodes labeled $(1, A)$. Now $C_2 \cap C_1 = \emptyset$; otherwise, player-1 can win by playing to the designated node labeled $(n-1, a)$ in C_1 again and create a unaccepted loop. From C_2 , again player-1 can play to a designated node labeled $(n-1, a)$, from where no matter what player-0 plays she can bring the play to $(1, A)$. She can again restrict the play to \mathcal{G}'_n and let the play settle to tSCC C_3 in player-0's MBWS graph. Again we can reason as before that $C_3 \cap C_2 = \emptyset$. Thus reasoning in this manner we get a sequence of tSCCs C_i for $i \geq 1$ such that $C_i \cap C_{i+1} = \emptyset$, and each C_i contains at least 2^{n-1} nodes labeled $(1, A)$. Therefore, the winning MBWS for player-0 in \mathcal{G}_n contains at least $2 * 2^{n-1}$ nodes labeled $(1, A)$, and hence player-0 needs at least 2^n amount of memory to win.

■

Lescow [30] shows another family of games to prove an exponential lower bound for the memory needed to win in Streett games. We are currently working on improving our memory-bound to a strategy size bound for Streett games, as well as looking to close the gap between the upper bound $n2^{s \log h}$ and the lower bound $n2^{O(s)}$.

3.9.3 Relationship between automata bounds and game bounds

The winning strategy for player-0 is a closed language *contained* in the game language, and so there is no reason to expect that a lower bound on translating a deterministic automaton into a minimum-pair deterministic chain automaton will be related to the lower bound on the memory required to win a game on an automaton with similar structural complexity. Translating a deterministic game automaton to a deterministic chain automaton furnishes both players with enough memory to win. Therefore, a strategy size lower bound on a particular game type also implies a lower bound for the translation of the game automaton into an equivalent chain automaton: but the reverse implication is not necessary, i.e., the strategy size upper bound may defy the translation lower bound. It is therefore interesting that there are many analogous results between translation of DOA into DCA and memory required to win in games on ω -automata.

Consider the game languages employed for proving the lower bounds for Muller games, and project down to player-1's alphabet, we get exactly the language family used to prove lower bounds for translating DMA into DBA and DCA (Section 2.13). Also, the bounds match up. However, the lower bound for the memory in general Streett games and the lower bound for translating DSA into DCA do not match up— 2^n for memory and $2^n n!$ for $DS \rightarrow DC$ —although the game language projected down to player-0's alphabet now is exactly the language used to prove the automata translation lower bound.

Two more noteworthy analogous results are:

1. Theorem 2.5.3 vs. Theorem 3.7.3. Negative union closed DOA can be realized as a DBA on the same transition structure, and negative union closed games

admit memory-less strategies for player-0.

2. Theorem 2.11.1 vs. Corollary 3.7.4. Positive and negative union closed DOA can be realized as DCA on the same transition structure, and positive and negative union closed games admit memory-less strategies for both players.

3.10 Summary

We considered a particular two-person game of perfect information played on an ω -automaton—the Gale Stewart game. These games admit a winner, and we characterized a special class of finite state strategies for the winner called memory-based strategies. These games can be equivalently viewed as a game played on a finite bipartite graph that considers both players symmetrically and makes the development of algorithms to decide the winner in the Gale Stewart game more convenient. We presented algorithms to decide the winner in Buchi, Rabin, and Streett games and to synthesize the winner’s strategy for an n -node h -pair Rabin (Streett) game in $O(n^{2k}h^k|E|)$ ($O(n^{2s}h^s|E|)$) time, where k (s) is the Rabin Index (Streett Index) of the game language, and E is the edge relation of the game graph.

Tree automata and games are closely related and can be translated between easily; trees accepted by automata correspond to winning strategies for player-0. Thus, our nonemptiness algorithms for tree automata have applications to decision problems of branching time temporal logics [14]. We also proved and conjectured lower bounds for the smallest strategy admitted by the player with Streett or Muller winning condition. In the next chapter we relax the constraint of perfect information and consider games of incomplete information.

Chapter 4

Two-person Games of Incomplete Information on ω -automata

4.1 Introduction

Consider the following two-person game played between two players: player-0 and player-1. In a single-step of the game player-0 chooses a letter from his alphabet $\Sigma_0 = \Sigma_0^{obs} \times \Sigma_0^{hid}$, followed by player-1 picking a letter from her alphabet $\Sigma_1 = \Sigma_1^{obs} \times \Sigma_1^{hid}$. These single-steps repeat ad infinitum to form a play of the game. Thus far the game defined is no different from the game defined in Chapter 3 except for the special form of the alphabet.

The following additional constraints define a modified Gale-Stewart game that we call a *Gale-Stewart game of incomplete information* or *partial observation*. Each letter in the players' alphabets is comprised of two parts: the observed and the hidden, indicated respectively by the superscripts *obs* and *hid*. A player only sees the observed part of his/her opponent's play: for instance, if player-0 picks $\sigma_0 = (\sigma_0^{obs}, \sigma_0^{hid})$, player-1 can only see σ_0^{obs} ; similarly player-0 only observes the *obs* part of player-1's "actions." Both the players are however privy to both the observed and hidden parts of their own actions.

A player may choose his/her next action based solely on the observed parts of the opposing player's play thus far. We define the Gale-Stewart incomplete information

game and the kind of strategies admissible, in Section 4.2. We also show that such games are not determined in Section 4.2. We consider the problem of deciding the winner and synthesizing the winner's strategy in Section 4.3; Section 4.3.1 consider's the computational complexity of the same problem. Section 4.4 considers two applications of GSI games: one in logic synthesis and the other in supervisory control. This chapter is summarized in Section 4.5.

4.2 Strategies for playing

The Gale-Stewart game of incomplete information (GSI game) is played on an ω -automaton, the game ω -automaton $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$, where $\Sigma_0 = \Sigma_0^{obs} \times \Sigma_0^{hid}$ and $\Sigma_1 = \Sigma_1^{obs} \times \Sigma_1^{hid}$. The game is played like a Gale-Stewart game of perfect information played on an ω -automaton; plays, winning plays, and partial plays are defined analogously to the case of the Gale-Stewart game of perfect information played on an ω -automaton. Strategies however are different.

A strategy ψ for player-0 is a function $\psi : (\Sigma_1^{obs})^* \rightarrow \Sigma_0$, or alternately a partial function $\psi : (\Sigma_0 \times \Sigma_1^{obs})^* \rightarrow \Sigma_0$ (we shall prefer the partial function form in the sequel), cueing his actions at every step. If player-0 plays according to his strategy ψ , the following play evolves:

$$\begin{aligned} & (\psi(\epsilon), \sigma_1^0 = (\sigma_1^{0obs}, \sigma_1^{0hid})) (\psi(\psi(\epsilon)\sigma_1^{0obs}), \sigma_1^1 = (\sigma_1^{1obs}, \sigma_1^{1hid})) \\ & (\psi(\psi(\epsilon)\sigma_1^{0obs}\psi(\psi(\epsilon)\sigma_1^{0obs})\sigma_1^{1obs}), \sigma_1^2 = (\sigma_1^{2obs}, \sigma_1^{2hid})) \dots \end{aligned} \quad (4.1)$$

Similarly a strategy τ for player-1 is a function $\tau : (\Sigma_0^{obs})^+ \rightarrow \Sigma_1$, or alternately and preferably a partial function $\tau : \Sigma_0^{obs}(\Sigma_1 \times \Sigma_0^{obs})^* \rightarrow \Sigma_1$.

A regular strategy for player-0 is a Moore FSM $S_0 = (Q_0, q_0^0, \Sigma_0, \Sigma_1^{obs}, \delta_0, \lambda_0)$. S_0 defines a closed language $\subseteq (\Sigma_0 \times \Sigma_1^{obs})^\omega$ as discussed in Section 3.2. In addition, we will find it useful in the sequel to associate a closed language $\hat{L}(S_0) \subseteq (\Sigma_0 \times \Sigma_1)^\omega$ with S_0 by basically replacing the label σ_0/σ_1^{obs} on an arc of the FSM S_0 by the labels $\sigma_0/(\sigma_1^{obs}, \sigma_1^{hid_1})$, $\sigma_0/(\sigma_1^{obs}, \sigma_1^{hid_2})$, \dots , $\sigma_0/(\sigma_1^{obs}, \sigma_1^{hid_{|\Sigma_1^{hid_1}|}})$, where $\Sigma_1^{hid} = \{\sigma_1^{hid_1}, \sigma_1^{hid_2}, \dots, \text{and } \sigma_1^{hid_{|\Sigma_1^{hid_1}|}}\}$. More formally:

Definition 4.2.1 Given a strategy FSM $S_0 = (Q_0, q_0^0, \Sigma_0, \Sigma_1^{obs}, \delta_0, \lambda_0)$ for player-0 is a GSI game $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$,

$$(\sigma_0^0, \sigma_1^0 = (\sigma_1^{0obs}, \sigma_1^{0hid}))(\sigma_0^1, \sigma_1^1 = (\sigma_1^{1obs}, \sigma_1^{1hid}))(\sigma_0^2, \sigma_1^2 = (\sigma_1^{2obs}, \sigma_1^{2hid})) \dots \in \hat{L}(S_0),$$

provided $\sigma_0^0 = \lambda_0(q_0^0)$ and for each $i > 0$, $\sigma_0^i = \lambda_0(\delta_0(q_0^0, \sigma_0^0 \sigma_1^{0obs} \sigma_0^1 \sigma_1^{1obs} \dots \sigma_0^{i-1} \sigma_1^{(i-1)obs}))$.

A regular strategy for player-1 is a Mealy FSM $S_1 = (Q_1, q_0^1, \Sigma_1, \Sigma_0^{obs}, \delta_1, \lambda_1)$. We associate a closed language $\hat{L}(S_1) \subseteq (\Sigma_0 \times \Sigma_1)^\omega$ as follows:

Definition 4.2.2 Given a strategy FSM $S_1 = (Q_1, q_0^1, \Sigma_1, \Sigma_0^{obs}, \delta_1, \lambda_1)$ for player-1 in a GSI game $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$, $(\sigma_0^0 = (\sigma_0^{0obs}, \sigma_0^{0hid}), \sigma_1^0)(\sigma_0^1 = (\sigma_0^{1obs}, \sigma_0^{1hid}), \sigma_1^1)(\sigma_0^2 = (\sigma_0^{2obs}, \sigma_0^{2hid}), \sigma_1^2) \dots \in \hat{L}(S_1)$ provided $\sigma_0^0 = \lambda_1(q_0^1, \sigma_0^{0obs})$ and for each $i > 0$, $\sigma_1^i = \lambda_1(q_0^1, \sigma_0^{0obs} \sigma_1^0 \sigma_0^{1obs} \sigma_1^1 \dots \sigma_0^{i-1obs})$.

Basically, we insert every possible letter from Σ_0^{hid} to get the labels on the arc of the corresponding ω -automaton.

A winning strategy is one that ensures a win for the player when he/she plays according to the strategy. For a GSI game played on a game automaton $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$, a regular strategy $S_0 = (Q_0, q_0^0, \Sigma_0, \Sigma_1^{obs}, \delta_0, \lambda_0)$ ($S_1 = (Q_1, q_0^1, \Sigma_1, \Sigma_0^{obs}, \delta_1, \lambda_1)$) is winning for player-0 (player-1) if $\hat{L}(S_0) \subseteq L(\mathcal{G})$ ($\hat{L}(S_1) \subseteq \overline{L(\mathcal{G})}$). As with Gale-Stewart games of perfect information, given a regular strategy for either player, there is a tracking relation between the strategy FSM and the game automaton, that is a simulation relation if the strategy is a winning one. We can thus also talk of memory-based strategies and memory-less strategies. Recall, in a memory-based strategy each state of the strategy FSM is tracked by exactly one state of the game automaton.

In a Gale-Stewart game of perfect information, since both players' actions are visible to each other, the concept of memory is useful to synthesize winning strategies. Whereas, in a GSI game, at the end of a one-step play (say) from the initial state, each player is (has to be) under the impression that the play could now be in any of a set of states (recall: the game automaton is known to both players but each can only observe the other's observable actions). We therefore add all unobservable actions of the opposing player to come up with the ω -automaton corresponding to a strategy

FSM; therefore memory based strategies would suffice only for the most trivial of GSI games.

Are all GSI games won?

Proposition 4.2.1 *GSI games are not determined.*

Proof: Consider the game shown in Figure 4.1. The alphabets of the players are:

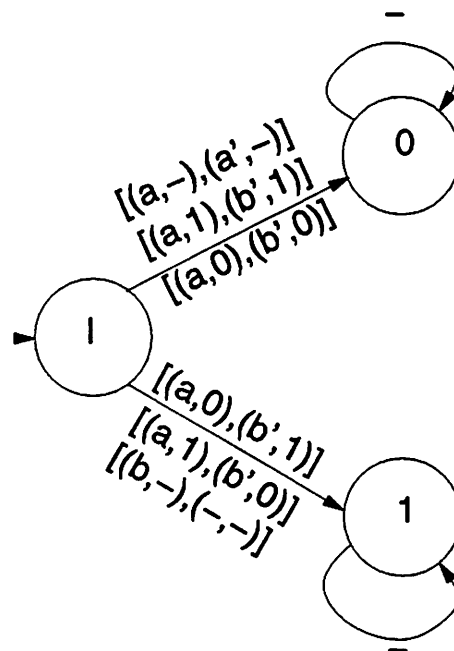


Figure 4.1: GSI game without a winner

$\Sigma_0^{obs} = \{a, b\}$, $\Sigma_0^{hid} = \{0, 1\}$, $\Sigma_1^{obs} = \{a', b'\}$, and $\Sigma_1^{hid} = \{0, 1\}$. The game automaton has three states labeled I , 0 , and 1 . We can impose a suitable acceptance condition (to result in an open or closed game language) to ensure that player-0 wins from the state labeled 0 and player-1 wins from the state labeled 1 .

Observe that, to win, from state I player-0 has to play a , and player-1 has to play b' , in the observable part of his/her action. It is then clear that neither player has a winning strategy. For instance, if player-0 plays $(a, 1)$ player-1 wins but only by playing $(b', 0)$, and if player-0 plays $(a, 0)$ player-1 wins only by playing $(b', 1)$; but,

player-1 can only observe that player-0 has played a . Similarly it can be argued that player-0 cannot guarantee a win either. ■

Thus there is not always a winner in a GSI game. How does one decide the winner if one exists, and synthesize his/her winning strategy?

4.3 Deciding GSI games on ω -automata

Consider an open GSI game $\mathcal{G} = \langle (Q, q_0, (\Sigma_0^{obs} \times \Sigma_0^{hid}) \times (\Sigma_1^{obs} \times \Sigma_1^{hid}), \delta), F \rangle$.

Definition 4.3.1 Let $\mathcal{G} = \langle (Q, q_0, (\Sigma_0^{obs} \times \Sigma_0^{hid}) \times (\Sigma_1^{obs} \times \Sigma_1^{hid}), \delta), \phi \rangle$ be a GSI game. Given $S \in Q$, $\sigma_0 \in \Sigma_0$, and $\sigma_1^{obs} \in \Sigma_1^{obs}$, the possible next states of q are defined to be

$$\hat{\delta}(S, \sigma_0, \sigma_1^{obs}) = \{q' | \exists q \in S \text{ and } \sigma_1^{hid} \in \Sigma_1^{hid} \text{ such that } q' = \delta(q, (\sigma_0, \sigma_1^{obs}, \sigma_1^{hid}))\}.$$

We compute the states from which player-0 may win, W_0 , as the sets of cardinality one among winning subsets of Q , \widehat{W}_0 , as follows. Any state in F is *a priori* a winning state for player-0, i.e., $F \subseteq W_0$; therefore $\{S | S \subseteq F\} \subseteq \widehat{W}_0$. Since player-0 cannot observe the hidden actions of player-1 he assumes that the play can be in any state in a subset of states. He can win from a state-subset S from which he can choose σ_0 such that every possible choice of σ_1^{obs} and every choice of σ_1^{hid} by player-1 puts the game in \widehat{W}_0 , i.e.,

$$S \in \widehat{W}_0 \text{ if for some } \sigma_0, \forall \sigma_1^{obs}, \hat{\delta}(S, \sigma_0, \sigma_1^{obs}) \subseteq \widehat{W}_0.$$

This basic idea can be generalized to other acceptance conditions as follows. Notice that $S_0 = (Q_0, q_0^0, \Sigma_0, \Sigma_1^{obs}, \delta_0, \lambda_0)$ is a winning strategy FSM for player-0 provided for every sequence:

$$(\sigma_0^0, \sigma_1^{0obs})(\sigma_0^1, \sigma_1^{1obs})(\sigma_0^2, \sigma_1^{2obs}) \dots \in L(S_0), \quad (4.2)$$

it is the case that for every sequence $\sigma_1^{0hid} \sigma_1^{1hid} \sigma_1^{2hid} \dots \in (\Sigma_1^{hid})^\omega$:

$$(\sigma_0^0, \sigma_1^{0obs}, \sigma_1^{0hid})(\sigma_0^1, \sigma_1^{1obs}, \sigma_1^{1hid})(\sigma_0^2, \sigma_1^{2obs}, \sigma_1^{2hid}) \dots \notin \overline{L(\mathcal{G})}. \quad (4.3)$$

(4.2) and (4.3) imply that player-0's objective is to produce a strategy FSM S_0 such that $L(S_0) \subseteq \overline{\Pi_{\Sigma_0 \times \Sigma_1^{obs}}(L(\mathcal{G}))}$.

We observe a simple lemma before proving the main theorem for the synthesis of strategies for GSI games.

Lemma 4.3.1 *Suppose $\mathcal{A} = \langle (Q, q_0, \Sigma_a \times \Sigma_b, \delta), \phi \rangle$ is a DOA, and $\Gamma_a = \overline{\Pi_{\Sigma_a}(L(\mathcal{A}))}$. Let $\sigma_a^0 \sigma_a^1 \sigma_a^2 \dots \in \Gamma_a^\omega$. Then for every sequence $\sigma_b^0 \sigma_b^1 \sigma_b^2 \dots \in \Sigma_b^\omega$, $(\sigma_a^0, \sigma_b^0)(\sigma_a^1, \sigma_b^1)(\sigma_a^2, \sigma_b^2) \dots \in L(\mathcal{A})$.*

Theorem 4.3.2 *Let $\mathcal{G} = (Q, q_0, (\Sigma_0 = \Sigma_0^{obs} \times \Sigma_0^{hid}) \times (\Sigma_1 = \Sigma_1^{obs} \times \Sigma_1^{hid}), \delta), \phi$ be a GSI game. Let $\Gamma = L(\mathcal{G})$, $\Gamma_0 = \overline{\Pi_{\Sigma_0 \times \Sigma_1^{obs}}(\Gamma)}$, and $\Gamma_1 = \Pi_{\Sigma_0^{obs} \times \Sigma_1}(\Gamma)$. Then player-0 wins \mathcal{G} iff he wins the Gale-Stewart game of perfect information with game language Γ_0 where he plays from Σ_0 and player-1 plays from Σ_1^{obs} ; player-1 wins \mathcal{G} iff she wins the Gale-Stewart game of perfect information with game language Γ_1 where player-0 plays from Σ_0^{obs} and player-1 plays from Σ_1 .*

Proof: First we consider Γ_0 and the case that player-0 might win. Assume that player-0 wins the GSP game with language Γ_0 and he has winning strategy $\psi : (\Sigma_0 \times \Sigma_1^{obs})^* \rightarrow \Sigma_0$. By Lemma 4.3.1, ψ is also a winning strategy for him in game \mathcal{G} .

Now assume that player-0 loses the GSP game with language Γ_0 ; call this game \mathcal{G}_0 . Let $\psi : (\Sigma_0 \times \Sigma_1^{obs})^* \rightarrow \Sigma_0$ be some strategy for him in \mathcal{G} . We will show that player-1 can beat player-0 in \mathcal{G} when he plays according to ψ . By the determinacy of GSP games, player-1 has a winning strategy, $\tau : \Sigma_0(\Sigma_1^{obs} \times \Sigma_0)^* \rightarrow \Sigma_1^{obs}$, to win in the GSP game \mathcal{G}_0 ; in particular, τ can beat ψ (in \mathcal{G}_0). Let the play that evolves in \mathcal{G}_0 when player-0 plays according to ψ and player-1 according to τ , be $\alpha = (\sigma_0^{0obs}, \sigma_0^{0hid}, \sigma_1^{0obs})(\sigma_0^{1obs}, \sigma_0^{1hid}, \sigma_1^{1obs})(\sigma_0^{2obs}, \sigma_0^{2hid}, \sigma_1^{2obs}) \dots$. Now $\alpha \in \overline{\Gamma_0} = \Pi_{\Sigma_0 \times \Sigma_1^{obs}}(\overline{\Gamma})$. Thus, there exists $\beta = \beta^0 \beta^1 \dots \in (\Sigma_1^{hid})^\omega$ such that:

$$\gamma = (\sigma_0^{0obs}, \sigma_0^{0hid}, \sigma_1^{0obs}, \beta^0)(\sigma_0^{1obs}, \sigma_0^{1hid}, \sigma_1^{1obs}, \beta^1)(\sigma_0^{2obs}, \sigma_0^{2hid}, \sigma_1^{2obs}, \beta^2) \dots \in \overline{\Gamma}. \quad (4.4)$$

We can thus define player-1's strategy $\tau' : \Sigma_0^{obs}(\Sigma_1 \times \Sigma_0^{obs})^* \rightarrow \Sigma_1$ as follows:

$$\begin{aligned} \tau'(\sigma_0^{0obs}) &= (\sigma_1^{0obs}, \beta^0) \\ \tau'(\sigma_0^{0obs} \sigma_1^{0obs} \beta^0 \sigma_0^{1obs}) &= (\sigma_1^{1obs}, \beta^1) \\ \tau'(\sigma_0^{0obs} \sigma_1^{0obs} \beta^0 \sigma_0^{1obs} \sigma_1^{1obs} \beta^1 \sigma_0^{2obs}) &= (\sigma_1^{2obs}, \beta^2) \end{aligned}$$

⋮

If player-1 plays according to τ' and player-0 according to ψ in \mathcal{G} the play γ will emerge and it will be a win for player-1 (by (4.4)). Therefore, we have shown that player-0 wins the game \mathcal{G} iff he wins the GSP game \mathcal{G}_0 .

We show next that player-1 wins the GSI game \mathcal{G} iff she wins the GSP game with language Γ_1 — \mathcal{G}_1 .

Let player-1 win the GSP game with language Γ_1 with strategy $\tau : \Sigma_0^{obs}(\Sigma_1 \times \Sigma_0^{obs})^* \rightarrow \Sigma_1$. When player-1 plays according to τ , let $(\sigma_0^{0obs}, \sigma_1^0)(\sigma_0^{1obs}, \sigma_1^1) \dots$ be an arbitrary play that evolves. Since τ is a winning strategy for player-1 in \mathcal{G}_1 ,

$$(\sigma_0^{0obs}, \sigma_1^0)(\sigma_0^{1obs}, \sigma_1^1) \dots \in \overline{\Gamma_1}$$

Then for all $\sigma_0^{0hid} \sigma_0^{1hid} \dots \in (\Sigma_0^{hid})^\omega$, by Lemma 4.3.1,

$$(\sigma_0^{0obs}, \sigma_0^{0hid}, \sigma_1^0)(\sigma_0^{1obs}, \sigma_0^{1hid}, \sigma_1^1) \dots \in \overline{\Gamma}.$$

Therefore τ is also a winning strategy for player-1 in the GSI game \mathcal{G} .

Now assume that player-1 loses the GSP game \mathcal{G}_1 (i.e., player-0 wins it). Let $\tau : \Sigma_0^{obs} \times (\Sigma_1 \times \Sigma_0^{obs})^* \rightarrow \Sigma_1$ be some strategy for her in \mathcal{G} . We show that player-0 can beat player-1 in \mathcal{G} when player-1 is playing according to τ . Let $\psi : (\Sigma_0^{obs} \times \Sigma_1)^* \rightarrow \Sigma_0^{obs}$ be a winning strategy for player-0 in \mathcal{G}_1 . In particular ψ beats τ (in \mathcal{G}_1). Let the play in \mathcal{G}_1 when player-0 and player-1 play respectively according to ψ and τ be $\alpha = (\sigma_0^{0obs}, \sigma_1^{0obs}, \sigma_1^{0hid})(\sigma_0^{1obs}, \sigma_1^{1obs}, \sigma_1^{1hid})(\sigma_0^{2obs}, \sigma_1^{2obs}, \sigma_1^{2hid}) \dots$. Now $\alpha \in \Gamma_1 = \Pi_{\Sigma_0^{obs} \times \Sigma_1}(\Gamma)$. Therefore there exists $\beta = \beta^0 \beta^1 \beta^2 \dots \in (\Sigma_0^{hid})^\omega$ such that:

$$\gamma = (\sigma_0^{0obs}, \beta^0, \sigma_1^{0obs}, \sigma_1^{0hid})(\sigma_0^{1obs}, \beta^1, \sigma_1^{1obs}, \sigma_1^{1hid})(\sigma_0^{2obs}, \beta^2, \sigma_1^{2obs}, \sigma_1^{2hid}) \dots \in \Gamma.$$

From this we can define player-0's strategy $\psi' : (\Sigma_1^{obs} \times \Sigma_0)^* \rightarrow \Sigma_0$ such that player-0 can employ ψ' to beat player-1 playing τ in \mathcal{G} .

If neither player wins, the game is obviously undetermined. ■

The game automata for the games with language Γ_0 and Γ_1 can be computed from the game automaton for \mathcal{G} .

4.3.1 Bounds

The crucial operation involved in the algorithm implied by Theorem 4.3.2 is a “co-determinization” step to complement a nondeterministic automaton to result in a deterministic game automaton. Closed and open GSI games can clearly be decided in EXPTIME, because determinization by the subset construction is exponential and deciding open and closed GSP games is in linear time.

For Rabin and Streett GSI games, Safra’s determinizations [42, 43] (for nondeterministic Buchi and Streett automata respectively), can be employed in conjunction with our constructions (DRA2DRCA or DRA2DSCA, see Chapter 2) to convert DRA or DSA into DCA to obtain DCA of size exponential in the size of the original game automata. This is possible because Safra’s determinization constructions result in a deterministic Rabin automaton with an exponential number of states but linear number of pairs, and our construction is exponential in the Rabin Index. Since deciding a GSP game on a chain automaton is in $\text{NP} \cap \text{Co-NP}$, deciding a Rabin or Streett GSI game turns out to be in $\text{NEXPTIME} \cap \text{Co-NEXPTIME}$, certainly in deterministic 2EXPTIME. How about lower bounds?

Proposition 4.3.3 *Deciding if player-0 wins a given Co-Buchi GSI game is PSPACE-hard.*

Proof: We transform NBA non-universality shown to be PSPACE-hard in [48]. Given a NBA $\mathcal{A} = \langle (Q, q_0, \Sigma, \delta), F \rangle$, the non-universality question is to test if $L(\mathcal{A}) \stackrel{?}{=} \Sigma^\omega$. The corresponding Co-Buchi GSI game is $\mathcal{G} = \langle (Q \sqcup \hat{f}, q_0, \Sigma_0^{obs} \times \Sigma_0^{hid} \times \Sigma_1^{obs} \times \Sigma_1^{hid}, \delta'), \neg(Q \sqcup \{\hat{f}\}) \setminus F \rangle$, where $\Sigma_0^{obs} = \Sigma$, $\Sigma_0^{hid} = \emptyset$, $\Sigma_1^{obs} = \emptyset$, and $\Sigma_1^{hid} = Q$;

$$\delta'(q, (\sigma_0^{obs}, \sigma_0^{hid}, \sigma_1^{obs}, q')) = \begin{cases} \hat{f} & \text{if } q = \hat{f} \\ q' & \text{else if } (q, \sigma_0^{obs}, q') \in \delta \\ \hat{f} & \text{otherwise} \end{cases}$$

Basically, we use a suitably large hidden alphabet for player-1 to simulate the nondeterministic transitions in \mathcal{A} . A play in \mathcal{G} is a won by player-0 if its infinity set does not contain a state from F . The game \mathcal{G} can be constructed in polynomial time.

Assume that for some $\sigma \in \Sigma^\omega$, $\sigma \notin L(\mathcal{A})$. If player-0 plays the sequence σ he is guaranteed a win, as every run of σ in \mathcal{A} eventually is contained in \overline{F} .

Similarly, a winning strategy for player-0 in \mathcal{G} yields a sequence $\sigma \notin L(\mathcal{A})$. Therefore $L(\mathcal{A}) \neq \Sigma^\omega$ iff player-0 wins \mathcal{G} . ■

Proposition 4.3.3 also implies that deciding Rabin or Streett GSI games in PSPACE-hard. In addition, although we have not shown that deciding open or closed GSI games is PSPACE-hard, we conjecture that:

Conjecture 4.3.4 *Deciding the winner in an open or closed GSI game is EXPTIME-complete.*

4.4 Applications of GSI games

In this section we consider two applications: one in logic synthesis and the other in supervisory control, where we apply the Gale-Stewart games of incomplete information that we defined and developed algorithms to decide. Both situations involve the following: there is a FSM (say) M and the problem is to design a FSM C that when “wired-up” with M satisfies a specification on their composed behavior.

4.4.1 Watanabe-Brayton FSM-component synthesis problem

The problem of synthesizing interacting FSMs was recently studied in [58, 1]. Figure 4.2 indicates the FSM to be synthesized, C , in dotted lines. The FSM M 's input wires are named x and u , supplied respectively from the environment and C . M 's outputs are y and v , where v is an input to C . The wires x , y , u , and v take on values over the sets Σ_x , Σ_y , Σ_u , and Σ_v respectively. The specification S on their combined behavior is a language $\subseteq (\Sigma_x \times \Sigma_y)^\omega$ given as an ω -automaton.

The composed behavior of M and C is defined as follows. Notice $L(M) \subseteq (\Sigma_x \times \Sigma_u \times \Sigma_v \times \Sigma_y)^\omega$ and $L(C) \subseteq (\Sigma_v \times \Sigma_u)^\omega$. When M and C are wired-up together as in Figure 4.2, their composed behavior has to agree on the common wires u and v . The composed machine $M \circ C$ denotes the language $L(M \circ C) \subseteq (\Sigma_x \times \Sigma_u \times \Sigma_v \times \Sigma_y)^\omega$, where

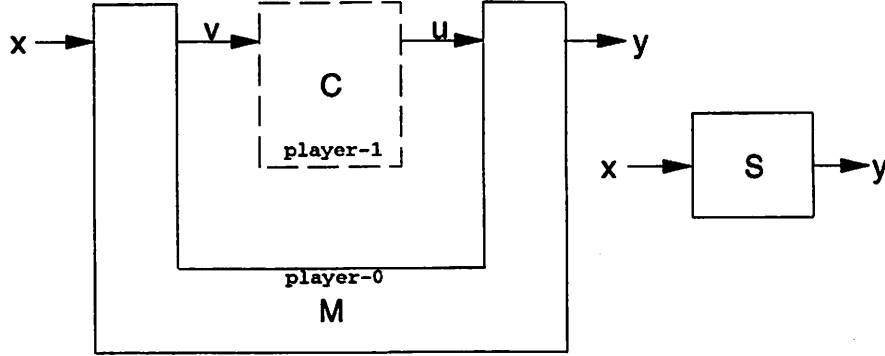


Figure 4.2: The Watanabe-Brayton component-FSM synthesis problem

$L(M \circ C) = L(M) \cap \hat{L}(C)$, and $\hat{L}(C) = \{\sigma \mid \Pi_{\Sigma_x \times \Sigma_v}(\sigma) \in L(C)\}$. The requirement on a *permissible* FSM C is that $\Pi_{\Sigma_x \times \Sigma_y}(L(M \circ C)) \subseteq L(S)$.

We show next that permissible FSMs C are regular winning strategies for player-1 in a GSI game.

Theorem 4.4.1 *Let $\Gamma = L(M) \cap \bar{\mathcal{L}}$, where $\mathcal{L} = \{\sigma \mid \sigma \in (\Sigma_x \times \Sigma_u \times \Sigma_v \times \Sigma_y)^\omega$ and $\Pi_{\Sigma_x \times \Sigma_y} \in L(S)\}$, be the game language for a GSI game with $\Sigma_0^{obs} = \Sigma_v$, $\Sigma_0^{hid} = \Sigma_x \times \Sigma_y$; $\Sigma_1^{obs} = \Sigma_u$ and $\Sigma_1^{hid} = \emptyset$. The set of winning strategy FSMs for player-1 in this GSI game is the set of permissible FSMs C for the Watanabe-Brayton FSM-component synthesis problem.*

Proof: Let C be a Mealy FSM that is a winning strategy for player-1 in the GSI game. Therefore, $\hat{L}(C) \subseteq \bar{\Gamma}$. Now $\bar{\Gamma} = \mathcal{L} \cup \overline{L(M)}$. Therefore (intersecting by $L(M)$ on both sides) $L(M) \cap \hat{L}(C) \subseteq \mathcal{L} \cap L(M)$. Therefore, $\Pi_{\Sigma_x \times \Sigma_y}(L(M) \cap \hat{L}(C)) \subseteq \Pi_{\Sigma_x \times \Sigma_y}(\mathcal{L} \cap L(M)) \subseteq \Pi_{\Sigma_x \times \Sigma_y}(\mathcal{L}) = L(S)$. Hence C is a permissible FSM.

Conversely, let C be a permissible FSM. Thus, $\Pi_{\Sigma_x \times \Sigma_y}(L(M) \cap \hat{L}(C)) \subseteq L(S)$. Therefore, $L(M) \cap \hat{L}(C) \subseteq \mathcal{L}$. Unioning $\overline{L(M)}$ on both sides, yields $\hat{L}(C) \subseteq \mathcal{L} \cup \overline{L(M)} = \bar{\Gamma}$, and therefore C is a winning strategy FSM for player-1. ■

Theorem 4.4.1 characterizes the E-machine [58, 1] as a game automaton on which a GSI game is played, and a permissible FSM is a winning strategy FSM for player-1.

4.4.2 Supervisory control of FSMs

In the supervisory control problem for FSMs, a plant FSM P has to be controlled by a controller FSM C so that the composed behavior of the plant and the controller has to satisfy the specification. The difference between the supervisory control problem and the FSM-component synthesis problem lies in which signals are observable.

The interconnection topology between the plant and the controller is shown in Figure 4.3. The controller C receives two inputs: x and y , supplied respectively by the environment and the plant. The controller's output u is the only input to the plant. An *admissible* controller C satisfies $\Pi_{\Sigma_x \times \Sigma_y}(\hat{L}(P) \times L(C)) \subseteq L(S)$. We can again pose the problem of finding an admissible controller as a GSI game.

Theorem 4.4.2 *Let $\Gamma = \hat{L}(P) \cap \overline{\hat{L}(S)}$ be the game language of a GSI game where $\Sigma_0^{obs} = \Sigma_x \times \Sigma_y$, $\Sigma_0^{hid} = \emptyset$, $\Sigma_1^{obs} = \Sigma_u$, and $\Sigma_1^{hid} = \emptyset$. Then, player-1's regular winning strategies are exactly the set of admissible controllers for the supervisory control problem of FSMs.*

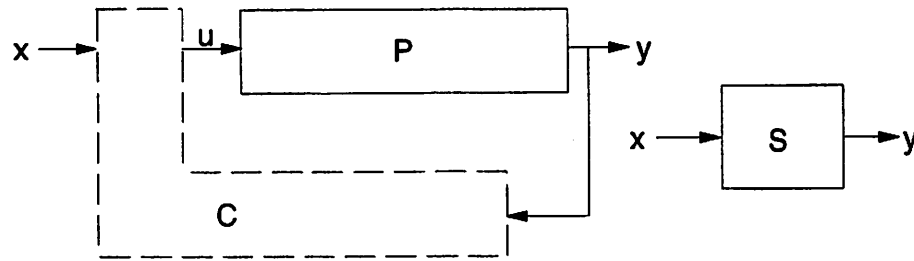


Figure 4.3: The FSM supervisory control problem

However, in the case of the supervisory control problem, neither player has any unobservable actions and we, in fact, have a GSP game.

In both the component-FSM synthesis and supervisory control problems, we can construct the game automaton for the game with language Γ by constructing a product automaton on the cartesian product of the state sets of the plant (or FSM M) and specification automata. Whereas the supervisory control problem results in a game of perfect information, the component-FSM synthesis problem results in a game of incomplete information (for player-1 at least). Therefore, we can check if the set

of admissible FSMs/controllers is non-empty in time linear in the size of the game automaton for the supervisory control problem [25], whereas it appears to take exponential time for the FSM-component synthesis problem.

4.5 Summary

We considered an incomplete information version of the Gale-Stewart game played on an ω -automaton—the Gale-Stewart game of incomplete information (GSI). At each step only the observable part of the choice of letter by a player is visible to his/her opponent, and “strategic” decisions may be based only on the opponent’s observable actions. GSI games are not determined. We converted the problem of deciding the winner in a GSI game to checking the winner in two GSP games. It is more difficult to decide GSI games than GSP games: while a Buchi GSP game can be decided in quadratic time, deciding a Buchi GSI game is PSPACE-hard. GSI games seem tailor made for modeling and solving the supervisory control and component FSM synthesis problems.

Two interesting questions on GSI games remain to be answered:

1. The conjectured EXPTIME lower bound for deciding open and closed GSI remains to be proved, and the lower bound for deciding other games also needs to be pinned down.
2. Minimum winning strategy size also needs to be studied. Even the case where the GSP game admits a memory-less strategy, we conjecture that the GSI game (of the same type) might need exponential-sized strategies, i.e., even a closed or open GSI game might require a strategy that is exponential in the size of the game automaton.

Chapter 5

Fair games on ω -automata; a synthesis example

5.1 Introduction

Formal verification of large systems necessitates employing simplified models of systems. These simplified models are usually created by hiding detail—abstraction—and the use of non-determinism. In order to better approximate the real behavior of the system being modeled restrictions on the admissible infinitary behavior are imposed, resulting in ω -automata. Imposing such restrictions on the infinitary behavior of systems captures semantic constraints on the systems such as fairness, progress, eventuality, justice, liveness, etc. [31]. Therefore, the restrictions on the infinitary behavior that arise in this context are called *fairness constraints*, since they constrain the behavior of the systems being modeled.

In order to be able to model the synthesis of components in a network of FSMs, some of which involve fairness constraints, we define a *fair two-person Gale Stewart game of perfect information* (FGSP game). In Section 5.2 we consider fair games on ω -automata, and in Section 5.3 we consider an example of synthesis illustrating many of the concepts considered in the thesis so far.

5.2 Fair games

A fair two-person Gale Stewart game of perfect information (FGSP) game is played on a game automaton $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$ just like a two-person Gale-Stewart game of perfect information. However winning plays and strategies are different.

In addition to the game automaton, there are two ω -regular languages, $L_{\phi_0} \subseteq \Sigma_0^\omega$ and $L_{\phi_1} \subseteq \Sigma_1^\omega$, respectively the fairness constraints for player-0 and player-1 given as DOA \mathcal{F}_0 and \mathcal{F}_1 : $L_{\phi_0} = L(\mathcal{F}_0)$ and $L_{\phi_1} = L(\mathcal{F}_1)$. Given a play $\pi = (\sigma_0^0, \sigma_1^0)(\sigma_0^1, \sigma_1^1)(\sigma_0^2, \sigma_1^2) \dots$, player-0 (player-1) is said to have *engaged in fair play* or *played fairly* provided $\Pi_{\Sigma_0} \in L_{\phi_0}$ ($\Pi_{\Sigma_1} \in L_{\phi_1}$). Play π is said to be a win for player-0 (player-1) if he (she) has engaged in fair play *and* either $\pi \in L(\mathcal{G})$ ($\pi \in \overline{L(\mathcal{G})}$) or player-1 (player-0) played unfairly. Thus if a player plays unfairly he/she cannot win.

A winning strategy for player-0 (player-1) is required to guarantee a win for him (her). That is, a winning strategy for (say) player-0 has to ensure that he always plays fairly, and either ensures a play in the game language or induces an unfair play from player-1.

5.2.1 Deciding FGSP games

The FGSP game can be solved as a GSP game. Player-0's winning plays are the set $\Gamma_0 = \hat{L}_{\phi_0} \cap (\Gamma \cup \overline{\hat{L}_{\phi_1}})$; player-1's winning plays are $\Gamma_1 = \hat{L}_{\phi_1} \cap (\overline{\Gamma} \cup \overline{\hat{L}_{\phi_0}})$, where for $i \in \{0, 1\}$, $\hat{L}_{\phi_i} = \{\sigma \in (\Sigma_0 \times \Sigma_1)^\omega \mid \Pi_{\Sigma_i} \in L_{\phi_i}\}$, and $\Gamma = L(\mathcal{G})$. Game automata representing the game languages Γ_0 and Γ_1 can be computed from the deterministic ω -automata for L_{ϕ_0} , L_{ϕ_1} , and the game automaton for the FGSP game, in polynomial time.

The inclusion relationship between Γ , $\overline{\Gamma}$, \hat{L}_{ϕ_0} , and \hat{L}_{ϕ_1} is depicted in Figure 5.1. The region shaded with horizontal lines is Γ_0 ; Γ_1 is shaded with vertical lines. The winning plays while being orthogonal do not necessarily exhaust the space of plays $(\Sigma_0 \times \Sigma_1)^\omega$, i.e., $\Gamma_0 \cup \Gamma_1 \neq (\Sigma_0 \times \Sigma_1)^\omega$. The interesting cases are when \hat{L}_{ϕ_0} and \hat{L}_{ϕ_1} are both non-empty; if they are both empty the game is not determined, as neither player can play fairly. If either fairness constraint is trivial, namely Σ_0^ω or Σ_1^ω , again

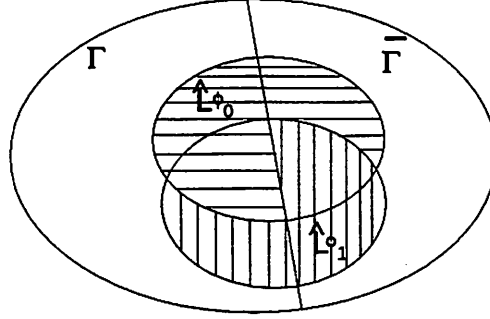


Figure 5.1: The game language and fairness constraints: containment relationship

$\Gamma_0 \cup \Gamma_1 = (\Sigma_0 \times \Sigma_1)^\omega$ and the game is determined. How about when L_{ϕ_0} and L_{ϕ_1} are both non-trivial and non-empty? $\overline{\Gamma_0} \cap \overline{\Gamma_1} = \overline{\hat{L}_{\phi_0}} \cap \overline{\hat{L}_{\phi_1}}$; is it possible that the players force each other to play unfairly?

5.2.2 (Non-)Determinacy of FGSP games

Theorem 5.2.1 *There exists a fair GSP game $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$, $L_{\phi_0} = L(\mathcal{F}_0 = \langle (Q_0, q_0^0, \Sigma_0, \delta_0), \phi_0 \rangle)$, and $L_{\phi_1} = L(\mathcal{F}_1 = \langle (Q_1, q_0^1, \Sigma_1, \delta_1), \phi_1 \rangle)$ that is non-trivial, i.e., $\Sigma_0^\omega \neq L_{\phi_0} \neq \emptyset$ and $\Sigma_1^\omega \neq L_{\phi_1} \neq \emptyset$, and that is not determined, i.e., where neither player has a winning strategy.*

Proof: Consider the game where: $\Sigma_0 = \{a, b\}$, $\Sigma_1 = \{a', b'\}$, $L_{\phi_0} = (a^*b)\{a, b\}^\omega$, $L_{\phi_1} = ((a')^*b')\{a', b'\}^\omega$, and $\overline{L(\mathcal{G})} = (a, a')^*(b, b')\{a, b\}^\omega$.

The game automaton for the game is shown in Figure 5.2.

First notice that player-0 playing fairly requires that he produce a “b” at some time. If he does that, player-1 can follow by showing a b' , thereby ensuring fairness and a win simultaneously. Therefore, if player-0 plays fairly he loses, so we can safely assume that he plays unfairly, i.e., plays a^ω .

Now, can player-1 play fairly thereby ensuring herself a win? Assume, player-1 does play fairly: at some time she produces a b' in response to player-0’s a . But any continuation of this partial play is in $L(\mathcal{G})$, and he can “restore fairness” to his play by putting out a b sometime, and thereby secure a win. Therefore player-1 will not play fairly either, she will play $(a')^\omega$.

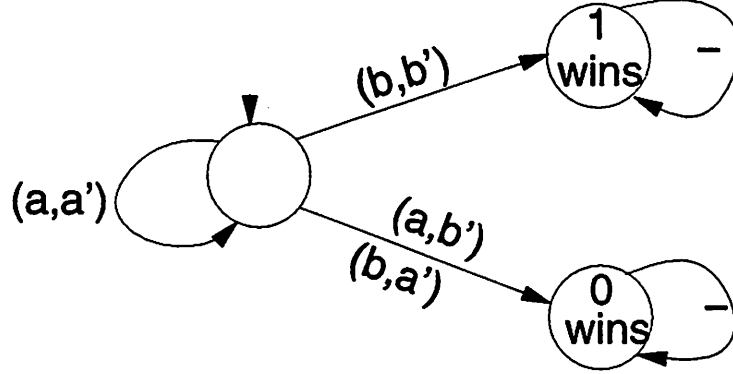


Figure 5.2: A fair GSP game with no winner

Thus both players are forced to play unfairly and neither can win. The game has no winner—it is not determined. ■

It is interesting that fair GSP games, although being games of perfect information are not determined. In the counter-example showing non-determinacy both players' fairness constraints are open languages.

Theorem 5.2.2 *Let $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$, $L_{\phi_0} = L(\mathcal{F}_0 = \langle (Q_0, q_0^0, \Sigma_0, \delta_0), \phi_0 \rangle)$, and $L_{\phi_1} = L(\mathcal{F}_1 = \langle (Q_1, q_0^1, \Sigma_1, \delta_1), \phi_1 \rangle)$ be a fair GSP game, where either L_{ϕ_0} or L_{ϕ_1} is a closed language. Then the game is determined.*

Proof: Assume that player-0 does not win. We will show that player-1 wins.

By the determinacy of GSP games, player-0 not winning implies that player-1 has a regular strategy ρ to ensure that the play is in $\overline{\Gamma_0} = \overline{\hat{L}_{\phi_0}} \cup (\overline{L(\mathcal{G})} \cap \hat{L}_{\phi_1})$. We modify ρ to yield a strategy ρ' for player-1 that will guarantee that the play will be in $\hat{L}_{\phi_1} \cap \overline{\Gamma_0}$.

When player-1 plays according to ρ let the partial play $\pi = (\sigma_0^0, \sigma_1^0)(\sigma_0^1, \sigma_1^1)(\sigma_0^2, \sigma_1^2) \dots (\sigma_0^i, \sigma_1^i) \dots$ arise. There are two possibilities:

- A. For some least $j \geq 0$, $\sigma_0^0 \sigma_1^0 \dots \sigma_0^j \sigma_1^j \in L_{\phi_0} = \emptyset$. Since L_{ϕ_1} is non-empty and ρ ensures that $\pi \in \overline{\Gamma_0}$, there exists $s_1^j s_1^{j+1} \dots$, a continuation of $\sigma_1^0 \sigma_1^1 \dots \sigma_1^{j-1}$ for player-1 such that $\sigma_1^0 \sigma_1^1 \dots \sigma_1^{j-1} s_1^j s_1^{j+1} \dots \in L_{\phi_1}$. ρ' plays this continuation resulting in the play

$$(\sigma_0^0, \sigma_1^0)(\sigma_0^1, \sigma_1^1)(\sigma_0^2, \sigma_1^2) \dots (\sigma_0^{j-1}, \sigma_1^{j-1})(\sigma_0^j, s_1^j)(\sigma_0^{j+1}, s_1^{j+1})(\sigma_0^{j+2}, s_1^{j+2}) \dots \in \hat{L}_{\phi_1}.$$

B. For every $j \geq 0$, $\sigma_0^0 \sigma_0^1 \dots \sigma_0^j \Sigma_0^\omega \cap L_{\phi_0} \neq \emptyset$. (Note that this do not imply that $\sigma_0^0 \sigma_0^1 \dots \sigma_0^j \dots \in L_{\phi_0}$, unless L_{ϕ_0} is a closed set.) Therefore:

$$(\sigma_0^0, \sigma_1^0)(\sigma_0^1, \sigma_1^1)(\sigma_0^2, \sigma_1^2) \dots (\sigma_0^j, \sigma_1^j)(\Sigma_0 \times \Sigma_1)^\omega \cap (\overline{L(\mathcal{G})} \cap \hat{L}_{\phi_1}) \neq \emptyset$$

which implies that:

$$(\sigma_0^0, \sigma_1^0)(\sigma_0^1, \sigma_1^1)(\sigma_0^2, \sigma_1^2) \dots (\sigma_0^j, \sigma_1^j)(\Sigma_0 \times \Sigma_1)^\omega \cap \hat{L}_{\phi_1} \neq \emptyset,$$

and this implies that:

$$\sigma_1^0 \sigma_1^1 \dots \sigma_1^j \Sigma_1^\omega \cap L_{\phi_1} \neq \emptyset.$$

($\sigma_1^0 \sigma_1^1 \dots \sigma_1^j \dots \in L_{\phi_1}$ if L_{ϕ_1} is a closed set.) In this case we set ρ' to play according to ρ .

The claim now is that if either L_{ϕ_0} or L_{ϕ_1} is a closed set, ρ' is a winning strategy for player-1. When player-1 plays according to ρ' let the play p evolve. Either:

1. $p \in \hat{L}_{\phi_0}$: In this case ρ' plays according to ρ and the play is a win for player-1 because $p \in \overline{L(\mathcal{G})} \cap \hat{L}_{\phi_1}$.
2. $p \notin \hat{L}_{\phi_0}$: In this case, either:
 - (a) Case A applies: $\Pi_{\Sigma_1}(p) \in L_{\phi_1}$ and p is a winning play for player-1. If L_{ϕ_0} is a closed language only this case applies.
 - (b) Case B applies: We may assume that L_{ϕ_1} is a closed set. Again $\Pi_{\Sigma_1}(p) \in L_{\phi_1}$ and p is a winning play for player-1.

■

5.3 Example

We consider the problem of the re-synthesis of a component in the “Traffic Light Controller” example distributed with the VIS formal verification system [19]. A little used farm road intersects with a multi-lane highway, and a traffic light controls the

traffic at the intersection. The light controller is implemented to maximize the amount of time the highway light remains green.

The architecture of the system is shown in Figure 5.3. The system is comprised of three modules: the timer, the highway (light) controller, and the farm-road (light) controller.

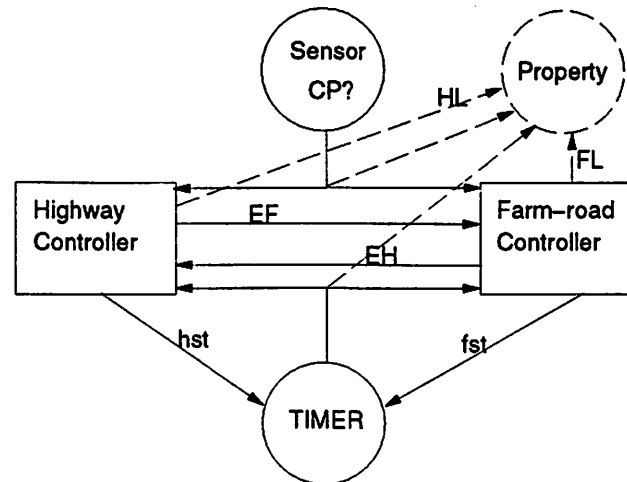


Figure 5.3: Architecture of the traffic light controller

The timer, Figure 5.4, serves to time the sojourns of the highway and farm-road lights through yellow and green. Rather than specify the timer as counting “ticks” it is modeled at a high level using non-determinism as composed of three states: START, SHORT, and LONG. Asserting *hst* (highway start timer) or *fst* (farm-road start timer) causes the timer to (reset to) START. If neither *hst* nor *fst* is asserted, a SHORT (LONG) duration nondeterministically expires from the START (SHORT) state. The same timer simultaneously serves both the highway and farm-road lights.

The deterministic FSMs representing the highway and farm-road traffic light controllers (HLC & FLC) are shown respectively in Figure 5.5 and Figure 5.6. The signals EF (enable-farm-light) and EH (enable-highway-light) outputs respectively of the HLC and FLC cause the FL and HL to transition from red to green. If a forward slash does not follow the input on a transition it means the output is not asserted.

The signal CP (car-present) indicates if a car is present at the traffic intersection on either the farm-road or the highway. The FL goes to red if either CP is asserted or

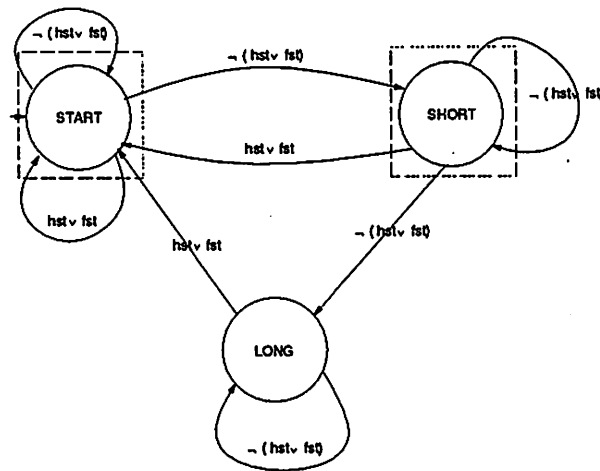


Figure 5.4: The timer

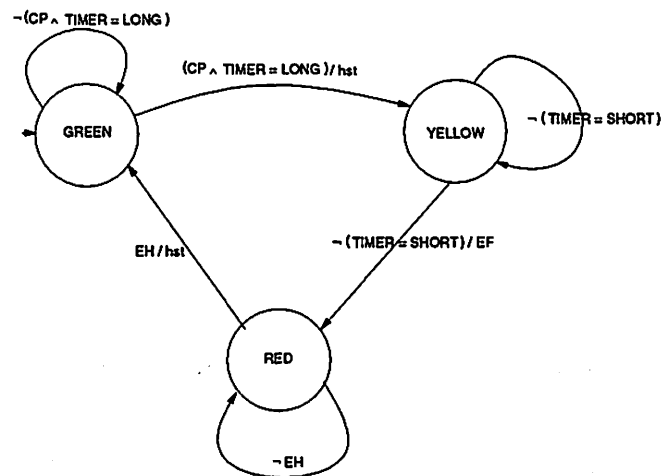


Figure 5.5: HLC: Highway traffic light controller FSM

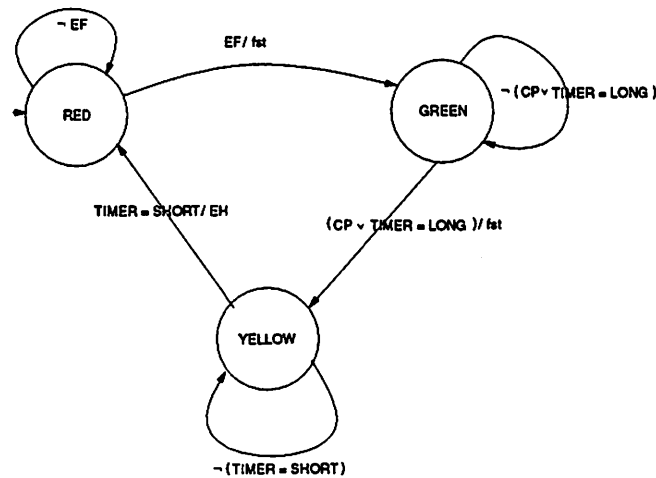


Figure 5.6: FLC: Farm road traffic light controller FSM

a “long” duration of time has elapsed, whereas the HL goes to red only if both CP is asserted and a long duration of time has elapsed. Both lights transition to red from yellow after a short duration of time has elapsed.

We consider next the problem of synthesizing permissible replacements for the highway control module, where permissible would be determined by the satisfaction of properties that we would require of the traffic light controller; three important ones proved in the VIS test suite are:

1. **Safety:** Both the highway light and the farm-road light are not green simultaneously. The deterministic ω -automaton (DOA) representing this property is shown in Figure 5.7; the acceptance condition is a single Streett pair: $(\emptyset, \{A\})$.
2. **Justice:** If a car is present on the farm road and the timer has expired then eventually the farm-road light should be green. The acceptance condition for the DOA in Figure 5.8 uses a single Streett pair: $(\{A\}, \{A, B\})$.
3. **Recurrence:** The highway light will be green repeatedly regardless of what happens on the farm-road. The acceptance condition for the DOA for this property uses one Streett pair: $(\{B\}, \{A, B\})$.

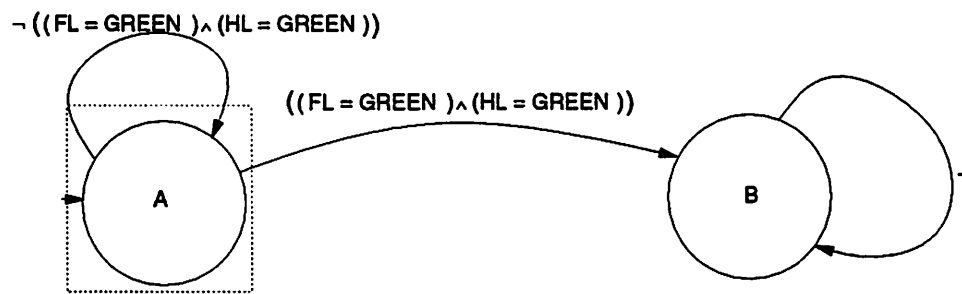


Figure 5.7: Safety: HL and FL not green simultaneously

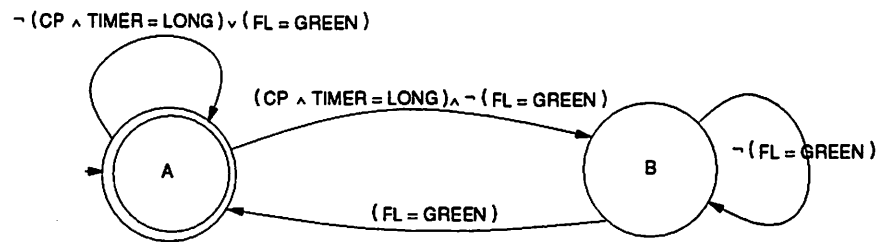


Figure 5.8: Justice: If car present and timer expired, eventually FL is green

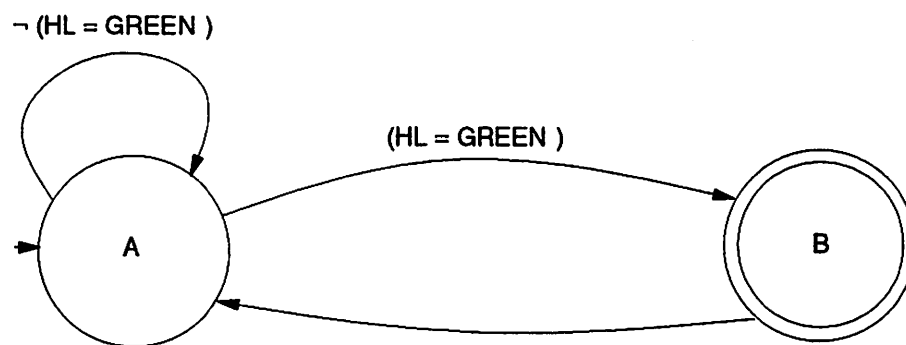


Figure 5.9: Recurrence: HL green infinitely often

The system as specified already fails some of the properties above. For instance the trace of the system: $(CP=NO, FL=RED, HL=GREEN, TIMER=START)(YES, RED, GREEN, SHORT)(YES, RED, GREEN, LONG)(YES, RED, YELLOW, START)^\omega$, fails the recurrence property as the timer forever stays in the state *START* although both *hst* and *fst* remain unasserted. To ameliorate this defect arising from the simplicity of our modeling, we impose the fairness constraint that if *hst* and *fst* remain unasserted continuously then timer cannot remain in the short or start state forever. For the system in consideration this requirement is effectively imposed by imposing a fairness constraint comprised of two Streett pairs: $(\emptyset, \overline{\{START\}})$, $(\emptyset, \overline{\{SHORT\}})$.

The set of permissible replacements for the highway controller module are computed as follows:

1. We compute a composite property P by constructing the intersection of the automata describing the safety, recurrence, and justice properties.

The composite property automaton (P) in Figure 5.10 is not complete; for instance, $HL = GREEN \wedge FL = GREEN$ would cause a transition to a “dead” state. The Streett pairs are: $(\emptyset, \{A, B, C, D\})$, $(\{A, C\}, \{A, B, C, D\})$, and $(\{C, D\}, \{A, B, C, D\})$.

2. We compute the synchronous product machine (M) of the modules timer and farm-road control to get the automaton shown in Figure 5.11.

Notice the state (*LONG, YELLOW*) is not reachable. The fairness constraint is composed of the two Streett pairs: $(\emptyset, \overline{\{START, -\}})$ and $(\emptyset, \overline{\{SHORT, -\}})$.

3. Following Section 4.4.1, we construct the automaton corresponding to the intersection of $\neg P$ and M . Permissible replacements for the highway control module are winning strategies for player-1 in the incomplete information game with game language $\overline{L(P)} \cap L(M)$ where $\Sigma_0^{obs} = \{CP, \neg CP\} \times \{EH, \neg EH\} \times \{START, SHORT, LONG\}$, $\Sigma_1^{obs} = \{EF, \neg EF\} \times \{hst, \neg hst\} \times \{RED, GREEN, YELLOW\}$, and all the other signals appear in Σ_0^{hid} .

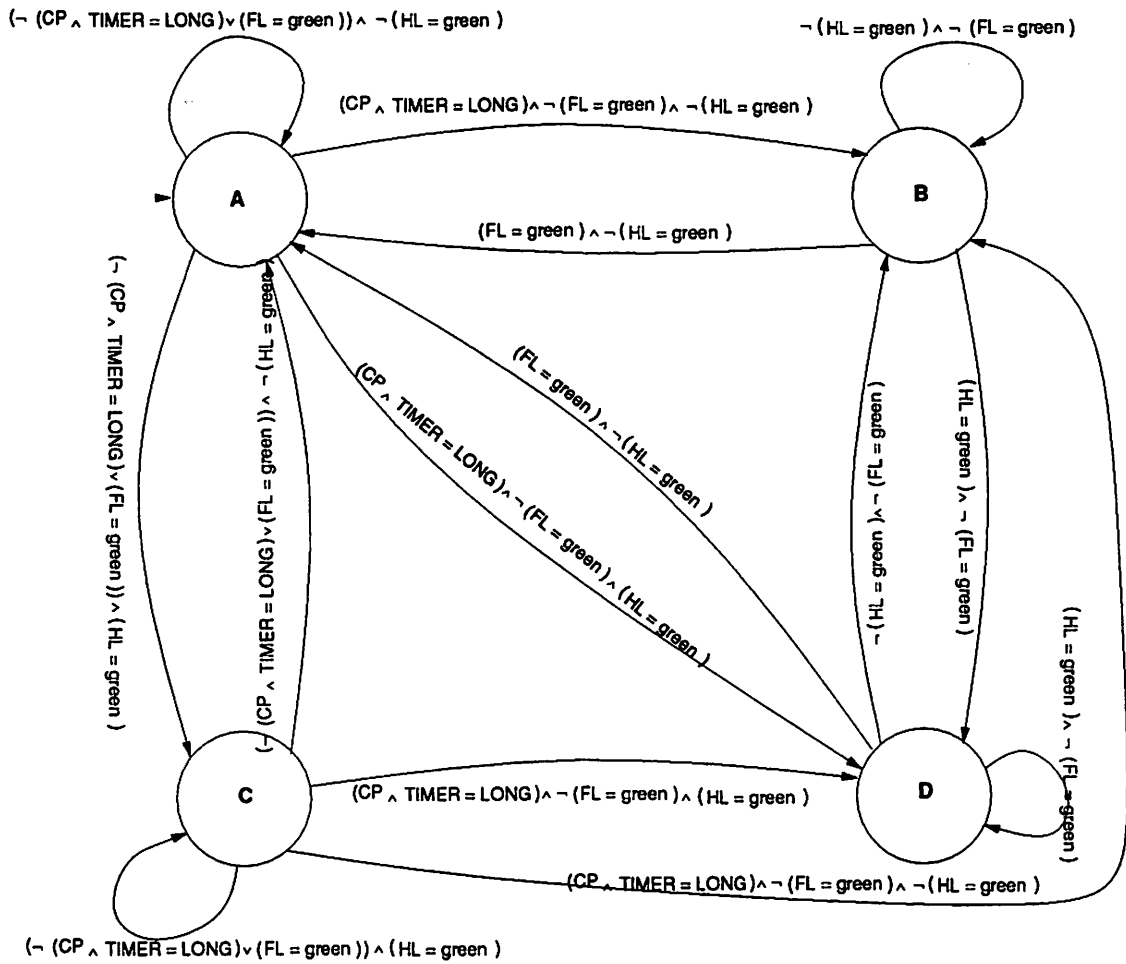


Figure 5.10: Composite property P

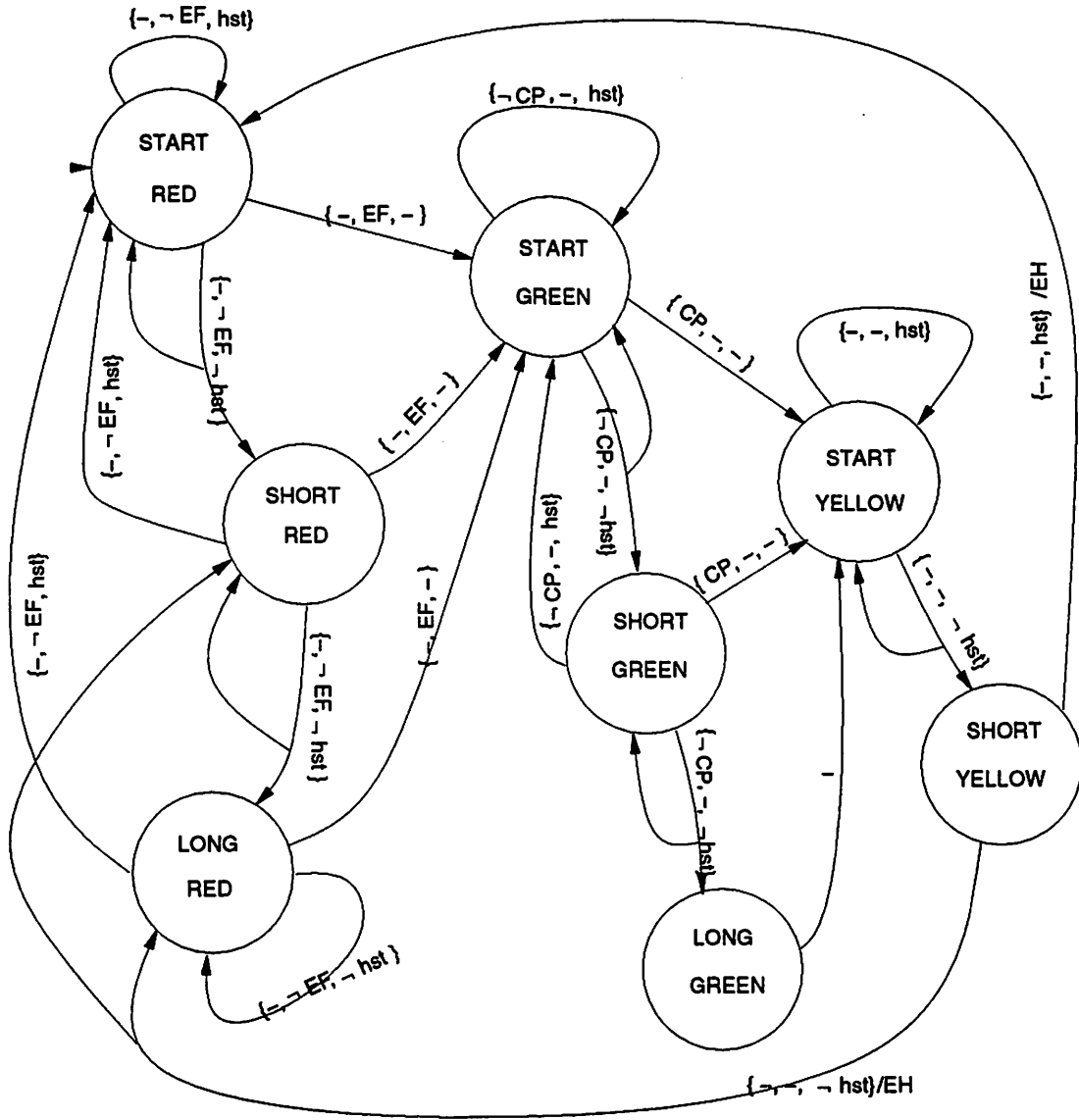


Figure 5.11: Product machine of timer and farm-road control modules

The deterministic game automaton for the incomplete information game discussed above is too large to draw on a single sheet. We instead “guess” some strategies and verify that they are indeed winning strategies using the VIS tool, and then successively augment the property.

The first module we consider is that in Figure 5.12.

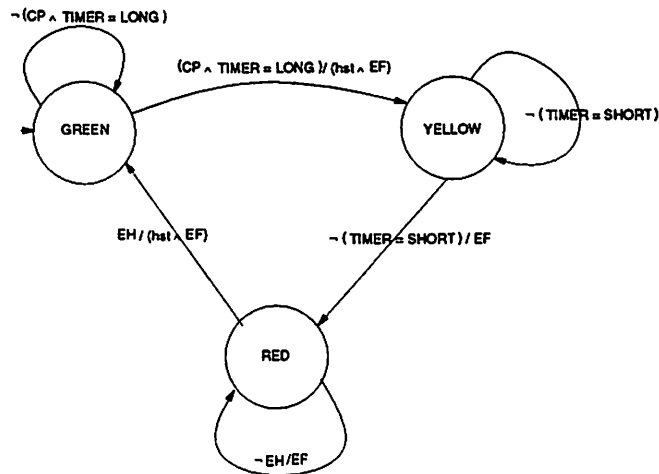


Figure 5.12: Highway Light Control Module I: HLCI

The difference between HLCI and the original highway light control module (HLC) is that EF is asserted on every transition. Using VIS it was verified that the original three properties proved were satisfied when HLCI replaced HLC. However, the system can now get into a state where FL is green and HL is yellow ((CP=NO, FL=RED, HL=GREEN, TIMER=START) (YES, RED, GREEN, SHORT) (YES, RED, GREEN, LONG) (YES, GREEN, YELLOW, START) ...), or FL and HL are yellow, both clearly unsafe states that need to be excluded.

We now add requirements that HL and FL are not simultaneously YELLOW, or one is YELLOW and other GREEN. After adding these requirements, a permissible FSM is shown in Figure 5.13.

Although HLCII satisfies all the properties we have required so far, it transitions from GREEN directly to RED. We further stipulate that the next state from GREEN is YELLOW or GREEN, from YELLOW is YELLOW or RED, and from RED is RED or GREEN. This in addition ensures that permissible FSMs are also “Moore in the

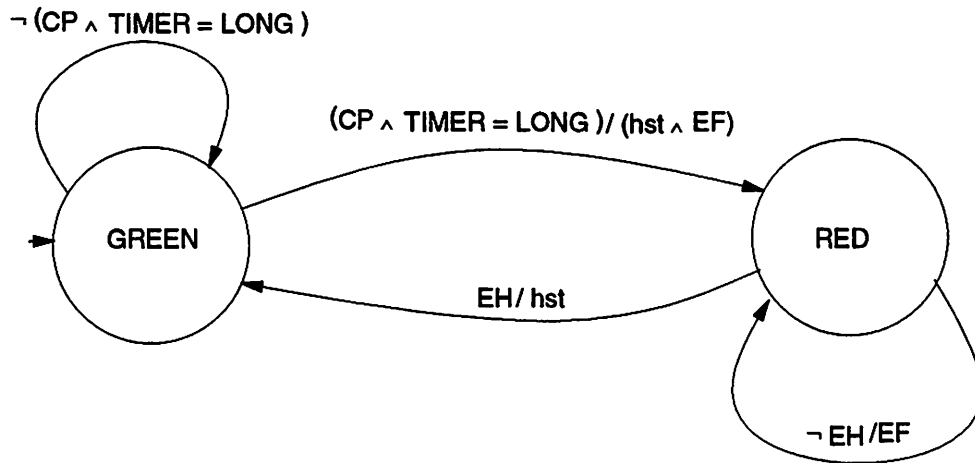


Figure 5.13: Highway Light Control Module II: HLCII

light.”

Figure 5.14 shows, HLCIII, a HLC FSM that conforms to all the properties we have required of the system so far.

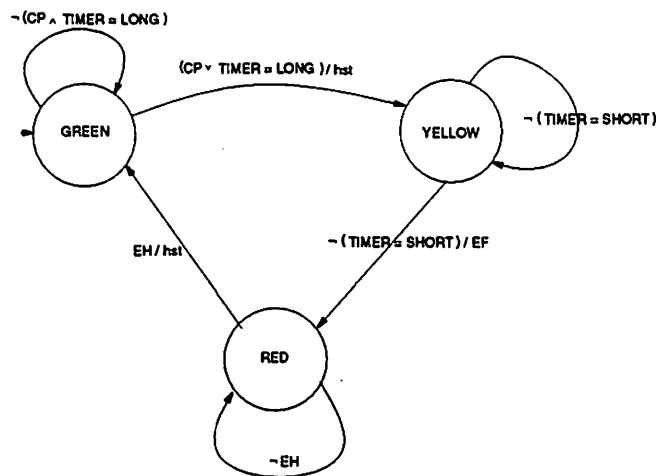


Figure 5.14: Highway Light Control Module III: HLCIII

Module HLCIII yields the intersection to farm-road under the same condition that the farm-road yields the intersection to the highway. However, we had originally set out to maximize the time the highway light remained green, although we didn't codify it into a property—it was more an informal requirement. Amongst the inputs,

EH is the signal from the farm-road light controller that it is transitioning to RED, so HLC's exit condition from GREEN has to be predicated on the signals CP and $TIMER$ only. Furthermore, the predicate has to imply the farm-road GREEN-exit condition, as we want to maximize the time the highway light turns green. Also, $CP \wedge TIMER = LONG$ has to cause an exit from GREEN (as required by the justice property). Putting these conditions together we are left with two possible exit conditions: $\neg CP \wedge TIMER = LONG$ or $CP \wedge TIMER = LONG$.

Therefore we have characterized the permissible replacements for the HLC module: modulo a choice of initial state of YELLOW or GREEN, and a choice of exit condition from GREEN (as discussed above), and a choice of sojourn-time through YELLOW (but we should additionally stipulate that is only a short duration), the original HLC is essentially the unique module up to isomorphism.

5.4 Summary

In this chapter we considered games where each player had a fairness constraint to adhere to. Although a game of perfect information, Fair Gale-Stewart games on ω -automata are not determined; however, if either player's fairness constraint is a closed language the game is determined. Fair games enable the modeling of synthesis of systems with fairness constraints on components.

We discussed the synthesis of a component in an example — the traffic light controller, involving both fairness constraints and incomplete information. A major problem is that of *completely* characterizing the property required of the permissible FSMs to replace a component in a network of interacting FSMs. The permissible FSMs we synthesized turned out to be minimum-state. In the next chapter we consider the problem of synthesizing minimum-state strategies.

Chapter 6

Minimum State Strategies

6.1 Introduction

A game automaton and the game language viewed as a synthesis requirement, represents a set of FSMs in the form of the different winning strategies the module to be synthesized has against the environment it is to function in, if the synthesis requirement is feasible. In choosing a particular winning strategy to be pushed through logic synthesis (to be implemented as a VLSI circuit), a minimum state strategy is considered a good starting point for deriving an efficient implementation.

In this chapter we consider the problem of synthesizing state-minimum strategies for Gale-Stewart games of perfect information on ω -automata (GSP games). Games of incomplete information on ω -automata as well as fair games can be translated to games of perfect information on ω -automata.

In Section 6.2 we consider the computational complexity of deriving minimum size memory-based as well as non-memory-based winning strategies. In Section 6.3 we argue that even when memory-less strategies are admitted the minimum state strategy may not lie in the space of memory-less strategies. In Section 6.4 we introduce subset FSMs, and employ them to compute the minimum state winning strategy FSMs for the player with the closed winning condition in Section 6.4.1. We investigate the synthesis of the smallest memory-based strategy in Section 6.5 and consider the synthesis of minimum sized non-memory-based strategies for non-closed winning conditions in

Section 6.6. Section 6.7 summarizes this chapter.

6.2 Computational complexity of deriving minimum-sized strategies

While deciding the winner in a Rabin game is NP-hard, we show next that synthesizing the smallest memory-less strategy for even a closed game is NP-hard.

The problem of deriving the smallest memory-less winning strategy (SMLS) for player-0 in a closed game may be stated as a decision problem as follows:

INSTANCE: A closed graph game¹ $\mathcal{G} = \langle G = (N = N_0 \sqcup N_1, E = E_0 \sqcup E_1), n_0, T \rangle$ and a positive integer k .

QUESTION: Does player-0 have a winning memory-less strategy containing $\leq k$ player-0 nodes?

Theorem 6.2.1 *SMLS is NP-complete.*

Proof: Given a ML-strategy for player-0 that includes n_0 , we can check that it is winning in time linear in the size of the strategy, putting SMLS in NP.

To show NP-hardness we reduce 3-SAT to SMLS. Let C be a 3-CNF formula with m clauses C_1, C_2, \dots, C_m , over n variables x_1, x_2, \dots, x_n . The corresponding SMLS instance has $3n + m + 1$ player-0 nodes labeled $r, C_1, C_2, \dots, C_m, 1, 2, \dots, n, x_1, x_2, \dots, x_n, \neg x_1, \neg x_2, \dots, \neg x_n$, and $2n + 1$ player-1 nodes labeled $r, x_1, x_2, \dots, x_n, \neg x_1, \neg x_2, \dots, \neg x_n$.

Each player-0 node with label from the set $\{r, x_1, x_2, \dots, x_n, \neg x_1, \neg x_2, \dots, \neg x_n\}$ has exactly one edge to the player-1 node with the same label. Each player-0 node labeled i , for $i \in \{1, 2, \dots, n\}$, has two edges out of it: one to x_i and the other to $\neg x_i$. A player-0 node labeled C_i has three edges, to the respective player-1 nodes labeled by the literals in C_i .

The player-1 node labeled r has edges to player-0 nodes with labels from the set $\{C_1, C_2, \dots, C_m, 1, 2, \dots, n\}$. Each player-1 node with label from the set $\{x_1, x_2, \dots,$

¹In this chapter we consider graph games with an initial player-0 node, $n_0 \in N_0$, from which the play begins.

$x_n, \neg x_1, \neg x_2, \dots, \neg x_n\}$ has exactly one edge to the player-0 node with the same label.

Example of the reduction: for the 3-CNF formula $(x_1 + x_2 + x_3)(\neg x_1 + x_4 + \neg x_3)(\neg x_4 + \neg x_2 + \neg x_3)$, the corresponding SMLS instance is shown in Figure 6.1.

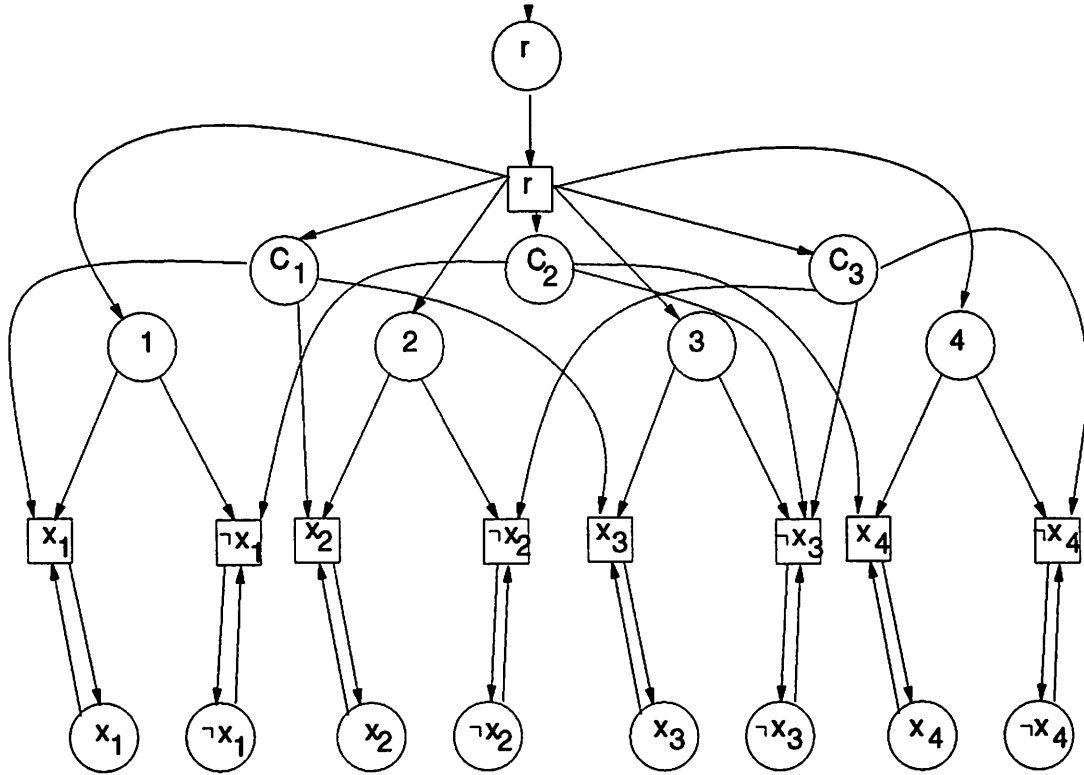


Figure 6.1: SMLS instance for 3-CNF instance $(x_1 + x_2 + x_3)(\neg x_1 + x_4 + \neg x_3)(\neg x_4 + \neg x_2 + \neg x_3)$

If $N = N_0 \sqcup N_1$ denotes the node set of the SMLS instance, we set $T = N$ and k to $2n + m + 1$. Clearly, the SMLS instance for a given 3-CNF instance can be constructed in polynomial time.

Given a 3-CNF formula that is satisfiable we can derive a ML winning strategy for player-0 with exactly $2n + m + 1$ player-0 nodes in the corresponding SMLS instance. The ML strategy is the subgraph of the game graph on the node set $N' = N'_0 \sqcup N'_1$, where the player-0 nodes are $N'_0 = \{r, C_1, C_2, \dots, C_m, 1, 2, \dots, n, l_1, l_2, \dots, l_n\}$, player-1 nodes are $N'_1 = \{r, l_1, l_2, \dots, l_n\}$, and l_1, l_2, \dots, l_n are the literals that constitute the satisfying assignment. The ML winning strategy is obtained by choosing from the player-0 node labeled $i \in \{1, 2, \dots, n\}$ the edge leading to the

player-1 node labeled by the literal l_i (in the satisfying assignment); and from player-0 node labeled C_j , $1 \leq j \leq m$, again an edge leading to a literal in the satisfying assignment. For example, in Figure 6.1, the satisfying assignment $\{x_1, \neg x_2, x_3, x_4\}$ induces a winning ML strategy with 12 player-0 nodes.

Similarly, it is easy to see that a winning ML strategy for player-0 with $2n + m + 1$ player-0 nodes induces a satisfying assignment for him.

Thus we have a polynomial time reduction from NP-complete 3-SAT problem to SMLS, showing that SMLS is NP-complete. ■

We can similarly also show that it is NP-hard to compute the smallest memory-less winning strategy for player-0 in an open game or for player-1 in either an open or a closed game.

As we might expect, synthesizing the smallest (minimum-state) strategy FSM (MSFSM) is at least as hard as synthesizing the smallest memory-less winning strategy:

Theorem 6.2.2 *Let $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), F \rangle$ be an open game, and k a positive integer. MSFMS is NP-complete, i.e., it is NP-complete to answer if player-1 has a winning strategy FSM with $\leq k$ states.*

Proof: There is a somewhat related problem in the literature, that of deriving the smallest (completely specified) deterministic Finite State Machine (DFSM) that is an extension of a given incompletely specified FSM. More precisely, given a sextuple $I = (Q, q_0, \Sigma_i, \Sigma_o, \delta, \lambda)$ where $\delta : Q \times \Sigma_i \rightarrow Q$ and $\lambda : Q \times \Sigma_i \rightarrow \Sigma_o$, respectively the transition and output function, are partial functions; the question is if δ and λ can be extended to total functions such that the resulting DFSM after state-minimization has $\leq k$ states, where k is a positive integer. This problem was proved to be NP-complete in [37].

Making all possible completions obviously results in a nondeterministic automaton. However, as we show below, we were able to modify the transformation in [37] from Graph k -colorability [18] to the reduction of an incompletely specified finite state machine, to result in a deterministic game automaton.

MSFSM is clearly in NP, as player-1 has a memory-less winning strategy in an open

game, and language containment between deterministic ω -automata for closed/open languages is in polynomial time.

The translation from Graph k -colorability to MSFSM closely follows that in [37]. The graph k -colorability problem is to determine if for a given graph² $G = (V, E)$ there is a (coloring) function $\psi : V \rightarrow \{1, \dots, k\}$ such that no adjacent vertices are assigned the same color, i.e., if $(u, v) \in E$ then $\psi(u) \neq \psi(v)$.

Given a graph $G = (V, E)$ we construct a game automaton $\mathcal{G} = \langle (Q, q_0, \Sigma_0 = V \times \Sigma_1 = \{0, 1\}, \delta), F \rangle$, where $Q = V \sqcup \{q_0, 0, 1, D\}$, $F = \{D\}$, and δ is defined next. For each $v \in V$:

$$\begin{aligned} \delta(q_0, (v, 0)) &= v, & \delta(q_0, (v, 1)) &= D, & \delta(0, (v, 0)) &= 0, \\ \delta(1, (v, 1)) &= 1, & \delta(0, (v, 1)) &= D, & \delta(1, (v, 0)) &= D, \\ \delta(v, (v, 0)) &= D, & \delta(v, (v, 1)) &= 1, & \delta(D, (v, 0)) &= D, \\ \delta(D, (v, 1)) &= D \end{aligned}$$

For each $(u, v) \in E$, $\delta(u, (v, 0)) = 0$, and $\delta(u, (v, 1)) = D$, and for $(u, v) \notin E$, $\delta(u, (v, 0)) = 0$, and $\delta(u, (v, 1)) = 1$.

Figure 6.2 illustrates the translation defined. Transitions from v_1, v_2, v_3 , and v_4 that are not shown are to the state labeled D .

It is clear that player-1 can keep the play within the set of states $Q \setminus \{D\}$ and win. At a node labeled $v \in V$, if player-0 plays u , player-1 has a choice of playing either 0 or 1 provided $(v, u) \notin E$; if $(v, u) \in E$ she is forced to play 0 to win.

The difference between our translation and that in [37], is that for $(v, u) \notin E$, in [37] $\delta(v, u)$ is unspecified, and hence can be assigned to any state, whereas we restrict the choice of next state to the set $\{0, 1\}$.

The resulting game automaton is clearly of size that is linear in the size of the graph. We claim that the graph G is k -colorable iff player-1 has a winning strategy FSM with at most $k + 3$ states (after state minimization). The proof is very similar to that of Theorem 1 in [37], and for completeness sake we briefly sketch out the main ideas.

²Assume without loss of generality that there are no vertices with “self-loops” or without any edges.

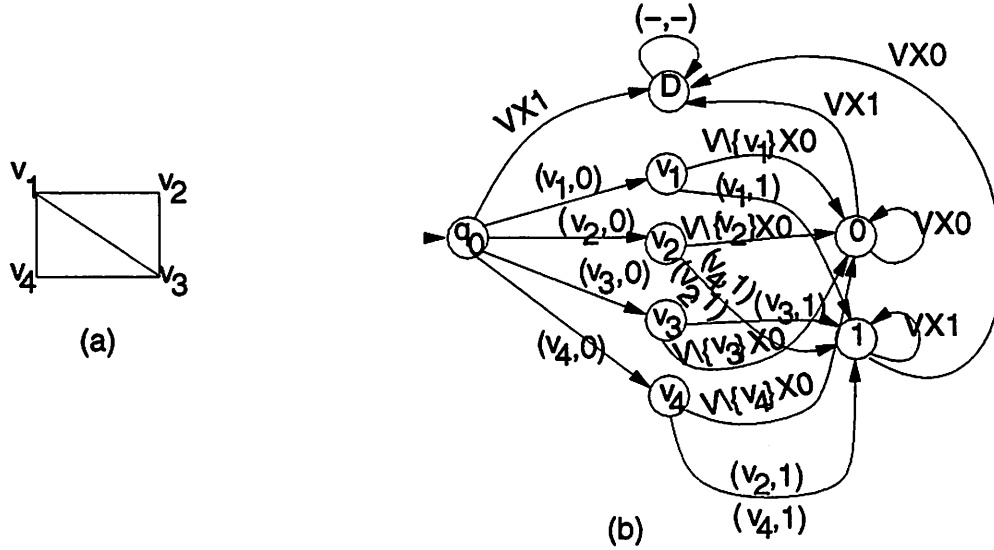


Figure 6.2: Transformation of k -colorability instance (a) to MSFSM instance (b)

Assume that G is k -colorable, by coloring function $\psi : V \rightarrow \{1, \dots, k\}$. Player-1 picks her winning strategy as follows. Recall, she has a choice of play from node v , on player-0 picking u only if $(v, u) \notin E$. She picks 0 iff for some v' : $(v', u) \in E$ and v' and v are colored the same, i.e., $\psi(v') = \psi(v)$. We claim now that for $v_1, v_2 \in V$, if $\psi(v_1) = \psi(v_2)$, states v_1 and v_2 are equivalent. Assume otherwise for contradiction, that for some $u \in V$, player-1 plays (say) 0 from v_1 and 1 from v_2 . Player-1 playing 0 from v_1 implies that either:

- (i) $(v_1, u) \in E$, or
- (ii) $(v_1, u) \notin E$ and $\exists v'$ such that $(v', u) \in E$ and $\psi(v_1) = \psi(v')$.

Player-1 playing 1 from v_2 implies that either:

- (a) $v_2 = u$, or
- (b) $v_2 \neq u$ and $\forall u'$ either $(u', u) \notin E$ or $\psi(u') \neq \psi(v_2)$.

It can be shown that in each of the four cases, namely (i)(a), (i)(b), (ii)(a), and (ii)(b), we get a contradiction. For instance, consider the case (ii)(b): let $u' = v'$.

Since $(v', u) \in E$, (b) implies that $\psi(u') \neq \psi(v_2)$. This contradicts the assumption ψ colors v_1 and v_2 the same, since it implies that $\psi(v_1) = \psi(v') = \psi(u') \neq \psi(v_2)$.

Therefore there are k -equivalence classes of states comprising the states from V , and the three additional states q_0 , 0, and 1, result in a strategy FSM for player-1 with no more than $k + 3$ states.

Conversely, we can also show that if player-1's winning strategy FSM has at most k states, G may be colored with $k - 3$ colors. First observe that from a state $v \in V$, player-1 has to play both 0 and 1 to win (if player-0 plays v , she has to play 1, and since no vertex is isolated, if player-0 plays u such that $(v, u) \in E$, she has to play 0). It follows that states q_0 , 0, and 1, are in equivalence classes by themselves in any winning strategy FSM for player-1. The $k - 3$ equivalence classes that comprise the states in V we claim determines the coloring, i.e., $\psi(v_1) = \psi(v_2)$ provided v_1 and v_2 are equivalent states. Assume for contradiction that ψ is not a proper coloring: that there are $v_1, v_2 \in V$ such that $\psi(v_1) = \psi(v_2)$ but $(v_1, v_2) \in E$. Since $\psi(v_1) = \psi(v_2)$, v_1 and v_2 are equivalent states. Since $(v_1, v_2) \in E$, player-1 has to play 0 from v_1 upon player-0 playing v_2 , whereas player-1 has to play 1 from v_2 upon player-0 playing v_2 , a contradiction. ■

Theorem 6.2.2 also shows that the problem of extracting a minimum state FSM from the E-machine [58] is NP-hard.

6.3 Insufficiency of the enumeration of ML-strategies

In an incompletely specified FSM one can enumerate the different completions of δ , minimize the resulting DFMSs, and so obtain the smallest DFMS that is a completion of the given incompletely specified FSM. Therefore, Theorem 6.2.2, also shows that it is NP-hard to determine the smallest strategy FSM that arises from considering memory-less strategies alone (and minimizing them). The question arises if a similar technique would work to obtain a state minimal strategy for GSP games.

Recall, every negative union closed winning condition admits a memory-less win-

ning strategy. If every smallest strategy FSM could be obtained by minimizing some memory-less strategy, or even if there is some memory-less winning strategy that could yield a minimum-state strategy upon state minimization, we could find a minimum state strategy FSM by enumerating memory-less winning strategies.

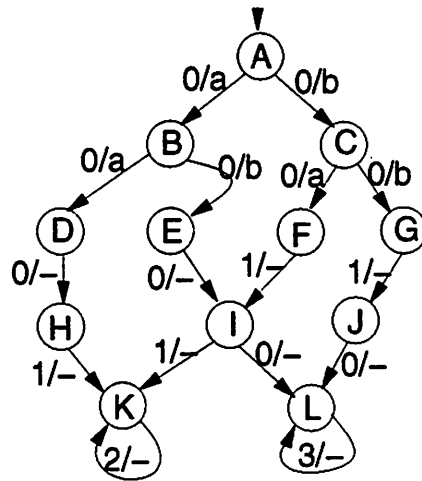


Figure 6.3: A game automaton where player-0 needs memory to yield a minimum-state strategy FSM

Unfortunately, Figure 6.3 illustrates that no memory-less strategy results in a minimum state winning strategy. Player-0's alphabet is $\{0, 1, 2, 3\}$; Player-1's alphabet is $\{a, b\}$. The game automaton may be completed by the addition of a dead state with a self-loop on every letter of the alphabet; all transitions not shown are to this dead state. The game language is a closed language: player-0's objective is to keep the play from getting to the dead state. The only state where he has a choice is the state labeled *I*. Picking (say) 1 from *I* results in a strategy FSM with 10 states, as states *D* and *E* become equivalent as do states *H* and *I*. Similarly choosing 0 from state *I* results in an FSM with 10 states. Whereas, if player-0 plays 1 from state *I* when the play comes from state *E* and 0 when the play comes from state *F*, the resulting FSM has 9 states, with the equivalence classes of states being $\{A\}$, $\{B\}$, $\{C\}$, $\{D, E\}$, $\{H, I\}$, $\{K\}$, $\{F, G\}$, $\{J, I\}$, and $\{L\}$.

Therefore, enumerating memory-less strategies as a means of deriving the state-minimum strategy FSM is not an option even for closed games. One possibility is

to enumerate strategy FSMs with at most a certain number of states determined by the upper bound on the size of the winning strategy. For instance, for Rabin games this upper bound is the number of states in the game automaton, whereas for Streett and Muller games the upper bound is exponential in the structural complexity of the game language.

However, enumeration by “brute force” doesn’t exploit the structure of the game automaton; can we do better?

6.4 FSMs on the state-subset space

Let $S_0 = (M_0, m_0^0, \Sigma_0, \Sigma_1, \delta_0, \lambda_0)$, be a strategy FSM for (say) player-0 in a game $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$. In general, a state $m \in M_0$ of the strategy FSM is tracked by a set of states $r(m) \subseteq Q$, i.e., $|r(m)|$ is possibly greater than 1, where $r \subseteq M_0 \times Q$ is the unique tracking relation between S_0 and \mathcal{G} . The given strategy S_0 is thus isomorphic to the FSM $(\{R \mid R = r(m) \text{ for some } m \in M_0\}, r(m_0^0), \Sigma_0, \Sigma_1, \delta'_0, \lambda'_0)$, where $\delta'_0(r(m), \sigma_1) = r(\delta_0(m, \sigma_1))$ and $\lambda'_0(r(m)) = \lambda_0(m)$.

Taking the cue from the above, we can enumerate strategy FSMs for a player in a game on an ω -automaton, by instead enumerating FSMs on the “subset space,” i.e., FSMs whose states are comprised of subsets of the states of the game automaton and whose transitions are “consistent” with the transition function of the game automaton.

Definition 6.4.1 *Given a transition structure $T = (Q, q_0, \Sigma, \delta)$ and $S \subseteq Q$ define $\hat{\delta}(S, \sigma) = \{q' \mid q' = \delta(q, \sigma) \text{ for some } q \in S\}$.*

Definition 6.4.2 *Given a game automaton $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$ a strategy FSM $\mathcal{M} = (M_0, m_0^0, \Sigma_0, \Sigma_1, \delta_0, \lambda_0)$ for player-0 is a **subset strategy FSM** provided:*

1. $M_0 \subseteq 2^Q$
2. $q_0 \in m_0^0$
3. For each $m \in M_0$, if $\delta_0(m, \sigma_1) = m'$ then $\hat{\delta}(m, (\lambda_0(m), \sigma_1)) \subseteq m'$

Definition 6.4.3 Given a game automaton $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$ a strategy FSM $\mathcal{M} = (M_1, m_0^1, \Sigma_1, \Sigma_0, \delta_1, \lambda_1)$ for player-1 is a **subset strategy FSM** provided:

1. $M_1 \subseteq 2^Q$
2. $q_0 \in m_0^1$
3. For each $m \in M_1$, if $\delta_1(m, (\sigma_0, \lambda_1(m, \sigma_0))) = m'$ then $\hat{\delta}(m, (\sigma_0, \lambda_1(m, \sigma_0))) \subseteq m'$

In addition, a subset-strategy FSM \mathcal{S} for player-0 (player-1) is a winning strategy FSM if $L(\mathcal{S}) \subseteq L(\mathcal{G})$ ($L(\mathcal{S}) \subseteq L(\bar{\mathcal{G}})$), where $\bar{\mathcal{G}} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \neg\phi \rangle$. This is equivalent to checking that the associated tracking relation between the strategy FSM and the game automaton is a simulation relation.

6.4.1 Finding the smallest strategy FSM for closed winning condition

In this section we pose the problem of finding the smallest winning strategy FSM for the player with the closed winning condition as the problem of finding an assignment to a set of Boolean variables that sets a minimum number of them to true and simultaneously satisfies a set of Boolean formulae.

Let $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), T \rangle$ be a closed game. Assume that the winning nodes for player-0, W_0 , contain q_0 , i.e., $q_0 \in W_0$. Given $C \subseteq W_0$, if for *some* $\sigma_0 \in \Sigma_0$, for *each* $\sigma_1 \in \Sigma_1$, $\hat{\delta}(C, \sigma_0, \sigma_1) \subseteq W_0$, we say C is a *viable* subset. Therefore, if $C \subseteq W_0$ is to be a state of a winning strategy subset-FSM for player-0, it is necessary that C be viable.

We introduce a Boolean variable x_C for each viable subset, C , of W_0 . If C is a state of a winning strategy FSM for player-0, then for *some* witness, σ_0 , of its viability, for each play $\sigma_1 \in \Sigma_1$ by player-1, a state that contains $\hat{\delta}(C, \sigma_0, \sigma_1)$ has to be present in the FSM. This is called the *closure constraint* and is expressed as a Boolean formula as follows:

$$\bigvee_{\sigma_0 \in A} (x_C \Rightarrow \bigwedge_{\sigma_1 \in \Sigma_1} \bigvee_{C' \in B} x_{C'}) \quad (6.1)$$

where $A = \{\sigma_0 | \sigma_0 \text{ is a "viability" witness for } C\}$ and $B = \{C' | \hat{\delta}(C, \sigma_0, \sigma_1) \subseteq C' \text{ and } C' \text{ is a viable subset}\}$.

Since the game starts from the initial state q_0 , the winning strategy FSM has to include some state that contains q_0 : the *initial state constraint* expressed as a Boolean formula is $\bigvee_{C' \in I} x_{C'}$, where $I = \{C' | q_0 \in C' \text{ and } C' \text{ is a viable subset}\}$.

A minimum state strategy FSM is obtained as an assignment to the Boolean variables for the viable subsets that sets a minimum number of them to true and satisfies the closure constraints and the initial state constraint. This optimization problem as cast is a 0 – 1 integer (non-linear) programming problem.

We next consider the problem of deriving the minimum state winning strategy subset FSM for player-1 in a open game, i.e., when player-1's winning plays form a closed set.

Let $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), F \rangle$ be an open game won by player-1, i.e., $q_0 \in W_1 \subseteq \bar{F}$. A subset of W_1 , $C \subseteq W_1$, is said to be viable if for each play $\sigma_0 \in \Sigma_0$ by player-0, player-1 can respond with $\sigma_1 \in \Sigma_1$ such that $\hat{\delta}(C, \sigma_0, \sigma_1) \subseteq W_1$. The closure constraints for player-1 take a form different from that for player-0. For each viable subset $C \subseteq W_1$ and $\sigma_0 \in \Sigma_0$, if C is a state of a winning subset strategy FSM for player-1, she has to be able to play some σ_1 (in response to σ_0) and transition to some viable subset; as a Boolean formula this is expressed as:

$$x_C \Rightarrow \bigvee_{\sigma_1 \in A} \bigvee_{C' \in B} x_{C'} \quad (6.2)$$

where $A = \{\sigma_1 | \hat{\delta}(C, \sigma_0, \sigma_1) \subseteq W_1\}$ and

$$B = \{C' | \hat{\delta}(C, \sigma_0, \sigma_1) \subseteq C' \text{ and } C' \text{ is a viable subset}\}.$$

Note that the closure constraint does not involve any conjunctions and there is one formula for each viable set and $\sigma_0 \in \Sigma_0$. The initial state constraint is identical to the player-0 case, and a minimum state strategy FSM for player-1 is derived as a satisfying assignment to the initial state constraint and closure constraints that sets a minimum number of viable subset-variables to true. Interestingly, for player-1 the optimization problem turns out to be a 0 – 1 integer *linear* programming problem, and also what is called a binate covering problem in the logic synthesis literature.

We defer arguing that the procedure defined above to solve a minimum-variable Boolean constraints satisfaction problem indeed does compute the minimum state winning strategy FSM until after we discuss computing the smallest memory-less strategies.

6.5 Computing the smallest memory-based winning strategy

If instead of having one Boolean variable for each subset of the winning set of states of the player with the closed winning condition, we have one Boolean variable for each state of the winning set, we can in a similar manner as for the subset FSM cast the problem of determining the smallest memory-less strategy as the problem of finding a satisfying assignment for a set of Boolean constraints with a minimum number of variables set to true. However, while the number of variables and constraints in the subset FSM case is possibly exponential in the cardinality of the winning set, it is linear for the problem of determining the smallest ML strategy.

The situation is a little different for non-closed winning conditions. Let's consider determining the smallest memory-less strategy for the winning player whose winning plays form an open language; let us consider an open game $\mathcal{G} = \langle (Q, q_0, \Sigma, \delta), F \rangle$ won by player-0 (from W_0).

It is no longer sufficient for player-0 to remain within W_0 to win; in addition, he has to visit a state in F at least once. If S is a winning memory-less strategy FSM for player-0, it follows that every infinite path in S starting from q_0 (more precisely the state in S simulated by q_0) has to pass through through F (states simulated by F).

We encode these constraints again as Boolean formulae. We introduce $|\Sigma_0| + 1$ Boolean variables for each state $q \in W_0$: a state variable x_q and *choice* variables $\{x_q^\sigma | \sigma \in \Sigma_0\}$.

The closure constraint takes on a different form:

$$\bigvee_{\sigma_0 \in A} (x_q \Rightarrow x_q^{\sigma_0} \bigwedge_{\sigma_1 \in \Sigma_1} \delta(q, (\sigma_0, \sigma_1))) \quad (6.3)$$

where $A = \{\sigma_0 \in \Sigma_0 \mid \text{forall } \sigma_1 \in \Sigma_1 \delta(q, \sigma_0, \sigma_1) \in W_0\}$. The choice variables help to track the choice of player-0 and hence paths in the strategy FSM. Besides the closure and initial state constraints we need to ensure that a visit to F is guaranteed. We impose this by disallowing infinite paths that steer clear of F . For each simple cycle C in the STG that does not contain a state from F , for each possible traversal of the simple cycle, we construct a Boolean cube by conjuncting the choice variables describing the edges along the traversal. The disjunction of the traversal cubes for a simple (negative) cycle forms the *cycle constraint*. The negation of the disjunction of the cycle constraints (one for each negative cycle) forms the *winning constraint*.

From a satisfying assignment for the initial state, closure, and winning constraints we may derive a memory-less winning strategy for player-0; and, a satisfying assignment that sets a minimum number of variables to true, yields a minimum-state memory-less winning strategy. This method could also be set up to work for deriving minimum size memory-less winning strategies for either player-0 or player-1 in games with more complicated winning condition such as Rabin, Streett, etc. Instead of considering only negative cycles we consider all negative SCSs in forming the winning constraint.

For the negative union closed winning condition, the smallest memory-based strategy is a memory-less strategy. Even for non-negative union closed conditions such as the Streett or Muller condition the procedure above will compute a memory-less strategy because we require that no negative SCS is traversed. Therefore if the player wins the game but not by a memory-less strategy the procedure above will not find a strategy. Also, for non-negative union closed winning conditions the minimum-state memory-based strategy may have fewer states than the minimum-state memory-less strategy. For all conditions, if the smallest memory-less strategy has (say) m states, the procedure above will compute a minimum satisfying assignment with $2m$ variables set to true; each state in the strategy will contribute 2 positive literals: one for the variable and one for the choice from the state.

For non-negative union closed winning conditions such as Streett and Muller, it remains to devise a procedure to find the smallest memory-based strategy other than by enumerating memory-based strategies.

6.6 Computing the smallest strategy FSM for non-closed winning conditions

Given a game $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$ and a memory-less strategy S for (say) player-0, it is easy to check if the strategy is winning: check if $L(\langle S, \neg\phi \rangle) = \emptyset$, where S specifies a transition structure on alphabet $\Sigma_0 \times \Sigma_1$ as discussed in Section 3.2. Similarly, it is easy to check if a memory-based strategy is a winning strategy. For a subset FSM for the winner with the closed winning condition, it is sufficient to check that the states are drawn from the winning set of nodes. Whereas, for a subset FSM strategy for a non-closed winning condition (open, Buchi, etc..) it is not possible to inherit the winning condition on the states of the subset FSM. We may however infer a winning condition by “unrolling” the subset FSM.

Definition 6.6.1 *Given a game automaton $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$ and a subset strategy FSM $\mathcal{M} = (M_0, m_0^0, \Sigma_0, \Sigma_1, \delta_0, \lambda_0)$ (for player-0), the **unraveled memory-based strategy FSM** corresponding to \mathcal{M} is a strategy FSM $\mathcal{M}' = (M'_0, m_0^{0'}, \Sigma_0, \Sigma_1, \delta'_0, \lambda'_0)$, where:*

1. *Each state $m = \{q_1, q_2, \dots, q_{|m|}\} \in M_0$, results in $|m|$ distinct states in M'_0 : $q_1^m, q_2^m, \dots, q_{|m|}^m$, and for $m_1, m_2 \in M_0$, $m_1 \neq m_2$, $\{q_1^{m_1}, q_2^{m_1}, \dots, q_{|m_1|}^{m_1}\} \cap \{q_1^{m_2}, q_2^{m_2}, \dots, q_{|m_2|}^{m_2}\} = \emptyset$;*
2. *The output function λ'_0 is consistent with λ_0 , i.e., $\lambda'_0(q_i^m) = \lambda_0(m)$ for all $q_i^m \in m \in M_0$;*
3. *The transition function δ'_0 is consistent with δ_0 and δ , i.e., $\delta'_0(q_i^m, \sigma_1) = r_j^{m'}$, where $\delta(q_i, (\lambda_0(m), \sigma_1)) = r_j$, $\delta_0(m, (\lambda_0(m), \sigma_1)) = m'$, $r_j^{m'} \in m'$, $q_i^m \in m$.*

Proposition 6.6.1 For \mathcal{G} , subset FSM \mathcal{M} , and unraveled FSM \mathcal{M}' as in Definition 6.6.1, the subset strategy FSM \mathcal{M} is winning provided $L(\langle \mathcal{M}', \neg\phi' \rangle)$, where ϕ' is such that for $S' \subseteq M'_0$, $S' \models \phi' \Leftrightarrow \Pi_Q(S') \models \phi$.

We can similarly define the unraveled memory-based strategy FSM corresponding to a subset strategy FSM for player-1, as well as check that it is winning for her by inheriting the winning condition on the states of the unraveled FSM.

We do not know how to pose the unraveling to check if a subset strategy FSM is winning, as a Boolean constraint on the Boolean variables for the subsets. In any event, for every strategy FSM and in particular the minimum-state strategy FSM, there is an isomorphic subset strategy FSM.

Several interesting questions arise as to the kinds of subsets that can comprise a *minimum* state winning strategy FSM:

1. Can there be two identical subsets with different output?
2. Can there be two subsets with identical output?
3. Can there be two subsets where one is contained in the other?

The answers to the above questions determine how many variables per subset we might need in enumerating subset strategy FSMs via a minimum variable Boolean constraints satisfaction approach. Recall, we introduced one variable per subset for the closed winning condition.

Proposition 6.6.2 Let $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$ be a game automaton and $\mathcal{M} = (M_0, m_0^0, \Sigma_1, \Sigma_0, \delta_0, \lambda_0)$ be a minimum-state subset winning strategy FSM (for player-0). Then, if $m_1, m_2 \in M_0$ are 2 states in \mathcal{M} , $m_1 \not\subseteq m_2$ and $m_2 \not\subseteq m_1$.

Proof: Assume for contradiction that (say) $m_1 \subseteq m_2$. It is easy to see that we can eliminate state m_1 from \mathcal{M} and redirect all transitions into m_1 to m_2 , thereby yielding a smaller strategy FSM. ■

Therefore the answer to the three questions posed above for winning strategies for closed games is no: given a winning subset strategy FSM, the maximal (in the

partial order induced by \subseteq) sets, choosing one set in case of more than one set with the same subset label, also comprise the state set of a winning subset-strategy FSM. Therefore, the minimum variable Boolean constraints satisfaction approach of Section 6.4.1 indeed does compute the smallest winning strategy FSM for the player with the closed winning condition.

We conjecture that for non-closed winning conditions, the answers to the three questions about the nature of sets comprising the state set of a minimum-state winning subset FSM is yes. We would need more than one Boolean variable per subset for the Boolean constraint based approach to minimum-state strategy synthesis; the number of variables per subset is certainly bounded from above by the upper bound on the size of the memory-based strategy admitted.

There are efficient practical solutions for binate covering problems [23, 10] that we can utilize to effectively synthesize minimum size memory-based strategies for games on ω -automata, as well as minimum-state strategy FSMs for the player with the closed winning condition. For non-closed winning conditions, our inability to translate the winning condition as a Boolean formula over the subset variables suggests that enumerating FSMs may be the only option to state-minimum strategy FSM synthesis.

6.7 Summary

In this chapter we studied the problem of synthesizing minimum-sized memory-based and non-memory-based strategies for games of perfect information on ω -automata. It is NP-hard to synthesize a minimum-state memory-based strategy or a non-memory-based strategy even for the player with a closed winning condition. The space of memory-based strategies is insufficient to synthesize a minimum-state strategy FSM even for winning conditions that admit memory-less strategies. FSMs with states comprised of subsets of the states of the game automaton—subset FSMs—provide a convenient abstraction to synthesize state minimal strategy FSMs. Minimum-state memory-less strategy synthesis can be cast as a minimum variable Boolean constraints satisfaction problem, as can minimum-state strategy FSM synthesis for the closed

winning condition. Previously, Watanabe [57] has used a similar approach for the extraction of a minimum state FSM from the E-machine.

In the next chapter we use subset FSMs to synthesize winning strategies that work with any state as initial state.

Chapter 7

Synthesizing uninitialized strategies

7.1 Introduction

Thus far in this thesis, a winning strategy has been a FSM with a unique initial state from which the FSM starts operating. In this chapter, we require that a winning strategy FSM ensure a win no matter which state of the strategy FSM it starts operating from. Given a game automaton, we consider the problem of deciding which player wins with an uninitialized strategy FSM—an FSM with no initial state.

In considering this problem we are motivated by the industrial design of sequential circuits where the state holding circuit elements—latches—have no reset circuitry and as a result may power-up in any state [46]. While an FSM with an initial state has a unique behavior, an uninitialized FSM's behavior depends on which state it starts in. Therefore there is some flexibility afforded in resynthesizing an uninitialized FSM. This flexibility can be cast as the set of winning uninitialized strategies for a Gale-Stewart game on an ω -automaton.

Sections 7.2 and 7.3 define uninitialized FSMs and subset FSMs. Sections 7.4 and 7.5 discuss the synthesis of uninitialized strategies for the player with the closed winning condition.

7.2 Uninitialized strategies

The game we consider is a two-person Gale-Stewart game of perfect information played on a (deterministic) game ω -automaton. The notion of plays, winning plays, and strategies is identical. Regular winning strategies are however differently defined: they are now FSMs with no initial state.

Definition 7.2.1 *An uninitialized Moore FSM is a quintuple $\mathcal{M} = (Q, \Sigma_o, \Sigma_i, \delta, \lambda)$, where:*

- Q is a finite set of states,
- Σ_o , a finite set, is the output alphabet,
- Σ_i , a finite set, is the input alphabet,
- $\delta : Q \times \Sigma_i \rightarrow Q$ is the transition function, and
- $\lambda : Q \rightarrow \Sigma_o$ is the output function.

An uninitialized Mealy FSM is similarly a quintuple $\mathcal{M} = (Q, \Sigma_o, \Sigma_i, \delta, \lambda)$, where the output function $\lambda : Q \times \Sigma_i \rightarrow \Sigma_o$ is dependent on the state and the input.

An uninitialized FSM $\mathcal{M} = (Q, \Sigma_o, \Sigma_i, \delta, \lambda)$ defines the closed language $\cup_{q \in Q} L((Q, q, \Sigma_o, \Sigma_i, \delta, \lambda))$.

Given a game automaton $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$, an uninitialized Moore strategy FSM $\mathcal{M}_0 = (M_0, \Sigma_0, \Sigma_1, \delta_0, \lambda_0)$ for player-0 is a *winning uninitialized strategy* for him provided for every $m_0^0 \in M_0$ the strategy FSM $S_0 = (M_0, m_0^0, \Sigma_0, \Sigma_1, \delta_0, \lambda_0)$ is a winning (initialized) strategy FSM for him, i.e., if $\cup_{m_0^0 \in M_0} L((M_0, m_0^0, \Sigma_0, \Sigma_1, \delta_0, \lambda_0)) \subseteq L(\mathcal{G})$. Similarly, an uninitialized Mealy strategy FSM $\mathcal{M}_1 = (M_1, \Sigma_1, \Sigma_0, \delta_1, \lambda_1)$ for player-1 is a winning uninitialized strategy for her provided for every $m_0^1 \in M_1$ the Mealy strategy FSM $S_1 = (M_1, m_0^1, \Sigma_1, \Sigma_0, \delta_1, \lambda_1)$ is a winning strategy for her.

7.3 Uninitialized Subset FSM

We will show that uninitialized winning strategies can be synthesized as subset FSMs.

Definition 7.3.1 Given a game automaton $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$ an uninitialized strategy FSM $\mathcal{M} = (M_0, \Sigma_0, \Sigma_1, \delta_0, \lambda_0)$ for player-0 is an uninitialized subset strategy FSM provided:

1. $M_0 \subseteq 2^Q$
2. For every $m \in M_0$, $q_0 \in m$
3. For each $m \in M_0$, if $\delta_0(m, \sigma_1) = m'$ then $\hat{\delta}(m, (\lambda_0(m), \sigma_1)) \subseteq m'$

Definition 7.3.2 Given a game automaton $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$ a strategy FSM $\mathcal{M} = (M_1, \Sigma_1, \Sigma_0, \delta_1, \lambda_1)$ for player-1 is an uninitialized subset strategy FSM provided:

1. $M_1 \subseteq 2^Q$
2. For every $m \in M_1$, $q_0 \in m$
3. For each $m \in M_1$, if $\delta_1(m, (\sigma_0, \lambda_1(m, \sigma_0))) = m'$ then $\hat{\delta}(m, (\sigma_0, \lambda_1(m, \sigma_0))) \subseteq m'$

Uninitialized subset FSMs are subset FSMs that not only do not have an initial state, but also require that the initial state of the game automaton q_0 be present in every state of the subset FSM.

For a game $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$, let $\mathcal{M}_0 = (M_0, \Sigma_0, \Sigma_1, \delta_0, \lambda_0)$ be an uninitialized strategy FSM for player-0. For every $m \in M_0$ there exists a unique tracking relation $r_0^m \subseteq M_0 \times Q$, between the states of the initialized FSM $\mathcal{M}_0^m = (M_0, m, \Sigma_0, \Sigma_1, \delta_0, \lambda_0)$ and \mathcal{G} . It follows that the relation

$$R_0 = \cup_{m \in M_0} r_0^m = \{(m, q) | (m, q) \in r_0^m \text{ for some } m \in M_0\}$$

is also a tracking relation. Therefore corresponding to the uninitialized strategy FSM \mathcal{M}_0 , the (unique) tracking relation R_0 defines an uninitialized subset FSM $\mathcal{S} = (S, \Sigma_0, \Sigma_1, \delta'_0, \lambda'_0)$ isomorphic to \mathcal{M}_0 on state set $S = \{R_0(m) | m \in M_0\}$, where $R_0(m) = \{q | (m, q) \in R_0(m)\}$, $\delta'_0(R_0(m), \sigma_1) = R_0(\delta_0(m, \sigma_1))$, and $\lambda'_0(R_0(m)) = \lambda_0(m)$. If M_0 is a winning strategy, the tracking relation R_0 is a simulation relation; the unraveled FSM corresponding to \mathcal{S} will contain $|M_0|$ “copies” of q_0 and any

strategy with initial state q_0 within the unraveled FSM will be a winning memory-based strategy FSM.

We can analogously show that player-1's uninitialized strategy FSMs are uninitialized Mealy subset FSMs.

7.4 Synthesizing uninitialized strategies for closed games

Consider a closed game $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), T \rangle$. Assume game \mathcal{G} is won by player-0 and his winning set is $W_0 \subseteq T$; $q_0 \in W_0$. We compute the set of subsets containing q_0 that can comprise the states of a winning uninitialized subset strategy FSM for player-0 iteratively as follows.

Consider subsets of W_0 with q_0 .

$$\rho_0^0 = \{S \mid S \subseteq W_0 \text{ and } q_0 \in S\}$$

For $i \geq 1$,

$$\rho_0^i = \{S \mid S \in \rho_0^{i-1} \text{ and } \exists \sigma_0 \in \Sigma_0 \text{ s.t. } \forall \sigma_1 \in \Sigma_1 \hat{\delta}(S, \sigma_0, \sigma_1) \subseteq S' \in \rho_0^{i-1}\}$$

For some I ,

$$\rho_0^{I+1} = \rho_0^I \stackrel{\text{def}}{=} \rho_0$$

Theorem 7.4.1 *Player-0 has an uninitialized winning strategy for closed game $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), T \rangle$ iff $\rho_0 \neq \emptyset$.*

Proof: If $\rho_0 \neq \emptyset$ it is clear that player-0 has an uninitialized winning subset strategy FSM.

Let $\mathcal{M}_0 = (M_0, \Sigma_0, \Sigma_1, \delta_0, \lambda_0)$ be a winning uninitialized strategy FSM for player-0. This is isomorphic to an uninitialized subset strategy FSM with state set $Q' = \{R_0(m) \mid m \in M_0\}$. It can be shown on induction on $i \geq 0$, that $Q' \subseteq \rho_0^i$. ■

7.4.1 E-machine for uninitialized strategies for player-0 in closed games

We can define a nondeterministic automaton on state set ρ_0 as follows. The alphabet of the automaton is $\Sigma_0 \times \Sigma_1$. For each state $q' \in \rho_0$ and each $\sigma_0 \in \Sigma_0$ such that for each $\sigma_1 \in \Sigma_1$, $\hat{\delta}(q', (\sigma_0, \sigma_1)) \subseteq S' \in \rho_0$, we have transitions from q' under (σ_0, σ_1) to every such state $S' \in \rho_0$; it is possible that there might be more than one allowed transition from a state for a given letter—a nondeterministic automaton. There is no initial state and the language consists of sequences with defined runs—a closed language.

This nondeterministic automaton can structurally simulate any winning uninitialized strategy FSM for player-0 for the closed game \mathcal{G} ; it is the E-machine [58] for uninitialized winning strategies for closed games won by player-0.

7.4.2 Uninitialized strategies for player-1 in open games

Assume that $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), F \rangle$ is an open game won by player-1, i.e., if $W_1 = \text{Closed}(\mathcal{G}, \bar{F}, 1)$, and $q_0 \in W_1 \subseteq \bar{F}$. We compute the subsets of W_1 containing q_0 that can comprise the states of a winning uninitialized subset strategy FSM for player-1 iteratively as follows. Start with:

$$\rho_1^0 = \{S \mid S \subseteq W_1 \text{ and } q_0 \in S\}$$

For $i \geq 1$,

$$\rho_1^i = \{S \mid S \in \rho_1^{i-1} \text{ and } \forall \sigma_0 \in \Sigma_0, \exists \sigma_1 \in \Sigma_1 \text{ s.t. } \hat{\delta}(S, \sigma_0, \sigma_1) \subseteq S' \in \rho_1^{i-1}\}$$

For some $I \geq 0$, $\rho_1^{I+1} = \rho_1^I \stackrel{\text{def}}{=} \rho_1$.

Theorem 7.4.2 *Player-1 has an uninitialized winning strategy in an open game $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), F \rangle$ iff $\rho_1 \neq \emptyset$.*

ρ_1 also defines an E-machine that structurally simulates all uninitialized winning strategies for player-1 in an open game.

7.5 Synthesis for safe replaceability

The synthesis problem for safe replaceability is: Given a (without loss of generality) Mealy uninitialized FSM $\mathcal{M} = (Q, \Sigma_1, \Sigma_0, \delta, \lambda)$ synthesize another Mealy uninitialized FSM $\mathcal{M}' = (Q', \Sigma_1, \Sigma_0, \delta', \lambda')$ such that $L(\mathcal{M}') \subseteq L(\mathcal{M})$; such an \mathcal{M}' is called a *safe replacement* for \mathcal{M} . The objective is to synthesize an \mathcal{M}' that has better characteristics such as area, delay, power, etc. The *flexibility* afforded for synthesizing \mathcal{M}' is the set of all \mathcal{M}' such that $L(\mathcal{M}') \subseteq L(\mathcal{M})$. Singhal [47] exploits a subset of the flexibility afforded and states that exploiting the complete flexibility via “an E-machine” is an open problem.

Since $L(\mathcal{M})$ is a closed language, and defines an open game won by player-1, valid safe replacements are exactly uninitialized strategy FSMs for player-1 for the open game with language $\overline{L(\mathcal{M})}$.

7.6 Non-closed games

In this section we consider the case that the winner’s plays do not form a closed set. Consider game $\mathcal{G} = \langle (Q, q_0, \Sigma_0 \times \Sigma_1, \delta), \phi \rangle$. Let W_0 be the nodes from which player-0 wins, and $q \in W_0$ (obviously, for a player to have an uninitialized winning strategy FSM he has to be able to win with an initialized strategy).

We make a few observations about the existence of an uninitialized winning strategy for player-0.

Let $L' = \bigcap_{q \in W_0} L(\langle (Q, q, \Sigma_0 \times \Sigma_1, \delta), \phi \wedge \neg \overline{W_0} \rangle)$. If player-0 wins the game with language L' , his winning initialized strategy FSM for such a game will also yield (by dropping the initial state) an uninitialized strategy FSM for him, because L' is the set of common winning plays from every node of W_0 .

If there exists a subset $S_0 \subseteq W_0$ such that $q_0 \in S_0$, and player-0 has a strategy to keep the play in S_0 forever, then again, if player-0 wins the game with language $L' = \bigcap_{q \in S_0} L(\langle (Q, q, \Sigma_0 \times \Sigma_1, \delta), \phi \wedge \neg \overline{S_0} \rangle)$, his winning strategy yields a winning uninitialized strategy FSM for him.

Finally, let

$$\rho_0^0 = \{S \mid S \subseteq W_0 \text{ and } q_0 \in S\}$$

For $i \geq 1$,

$$Q_{i-1} = \{q \mid q \in S \in \rho_0^{i-1}\}$$

$$\rho_0^i = \{S \mid S \in \rho_0^i \text{ and player-0 wins the game with language}$$

$$\bigcap_{q \in S} L(\langle (Q, q, \Sigma_0 \times \Sigma_1, \delta), \phi \wedge \neg \overline{Q_{i-1}} \rangle)\}$$

For some I ,

$$\rho_0^{I+1} = \rho_0^I = \rho_0$$

The subsets in ρ_0 are candidate subsets to comprise the state space of a winning uninitialized subset strategy FSM for player-0. Therefore, if $\rho_0 = \emptyset$ player-0 does not win with an uninitialized strategy. If $\rho_0 \neq \emptyset$, however, unless $L(\mathcal{G})$ is closed, player-0 does not necessarily have a winning uninitialized strategy FSM.

Subsets not in ρ_0 cannot participate in a winning subset strategy FSM. If we can bound the size of a subset FSM, enumerating subset FSMs of size no larger than the bound would yield an algorithm to decide the existence of a winning uninitialized subset FSM. Other than for closed games, we have not been able to bound the size of subset FSMs. While for the closed winning condition the answer to the three questions on Page 151 is no (Prop. 6.6.2 holds), we conjecture the answers for other winning conditions is yes. Unlike for initialized strategies amongst which are memory-based strategies that always exist and hence give a resulting upper bound on the subset strategy FSM size, for uninitialized strategies we do not have such a result.

We conjecture therefore that deciding the existence of uninitialized winning strategies would not be possible in general: the problem is undecidable.

7.7 Determinacy

The issue of determinacy is however easily settled: even closed games are not determined when we require uninitialized strategies.

Proposition 7.7.1 *There exist closed games where neither player has an uninitialized winning strategy FSM.*

Proof:

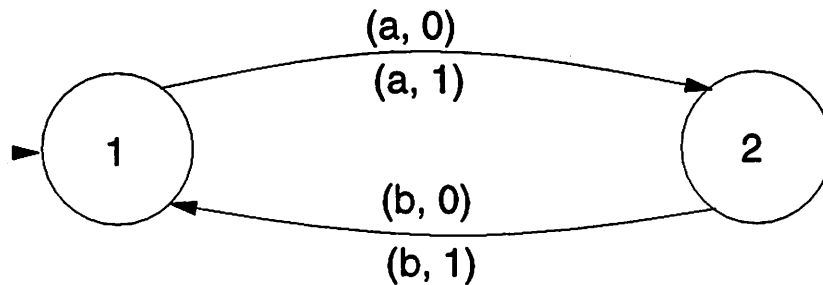


Figure 7.1: Closed game where neither player has an uninitialized winning strategy

Figure 7.1 shows a closed game won by player-0, $\Sigma_0 = \{a, b\}$, $\Sigma_1 = \{0, 1\}$; his objective is to keep the play in the two states indicated. However, he does not have an uninitialized winning strategy: there is no common winning choice that works for him from both states 1 and 2. ■

7.8 Summary

In this chapter we addressed the problem of synthesizing uninitialized strategies for two-person Gale-Stewart games of perfect information on ω -automata. FSMs with no initial state are gaining industrial use and we have addressed the property synthesis problem in this context. We presented an algorithm to decide if the player with the closed winning condition has an uninitialized winning strategy. This procedure also facilitates the exploitation of the complete flexibility available for synthesis for safe replaceability: we provided the E-machine construction for synthesis for safe replaceability.

Even closed games are not determined when we require uninitialized strategies. We gave some sufficient conditions for deciding the existence of uninitialized strategies for the player with the non-closed winning condition, and conjectured that this problem

is undecidable in general. Settling this conjecture remains an important item for future work.

Chapter 8

Conclusion

In this dissertation we related the structural complexity of deterministic ω -automata (DOA) to the synthesis of Finite State Machines (FSMs) from properties specified by deterministic ω -automata, as well as studied games that model synthesis problems. We summarize the contributions of this thesis and indicate some avenues for ongoing and future work.

The main results of Chapter 2 are summarized in Table 8.1. The second and third columns indicate the complexity of translation: for instance, we have $DS(n, h) \rightarrow DC(n2^{O(m \log h)}, k)$, where n is the number of states, h is the number of pairs and k (s) is the Rabin Index (Streett Index) of the DSA, and $m = \min(k, s)$. The optimality of all the translations were established by showing matching lower bounds. The column, RI?, indicates the complexity of determining the Rabin Index for the automaton type indexed by the row: for instance, if a DOA is both positive and negative union closed

	DB	DC	RI?	MinWit
DB(n)	n	$(n, 1)$	P	P
DR(n, h)	n	$(n2^{O(m \log h)}, k)$	NP-hard	P
DS(n, h)	nh	$(n2^{O(m \log h)}, k)$	NP-hard	P
DC(n, h)	n	(n, k)	P	P
DM(n, h)	$2^{O(n)}$	$2^{O(n \log n)}$	P	NP-hard

Table 8.1: Chapter 2 results' summary

such as a DCA, its Rabin Index can be determined in polynomial time. The last column, MinWit, is the complexity of finding the minimum length witness for the non-emptiness of the (nondeterministic) automaton type indexed by the row.

A language that is realizable as a DBA or a Co-DBA has a unique minimum state automaton. State minimization of DBA and other DOA remains an important open problem in the area of ω -automata.

In chapter 3, we addressed Gale-Stewart games of perfect information on ω -automata (GSP games). These games admit a winner, and we synthesized memory-based strategies by developing algorithms on game graphs. We presented algorithms to decide the winner in Buchi, Rabin, and Streett games and synthesize the winner's strategy for an n -node h -pair Rabin (Streett) game in $O(n^{2k}h^k|E|)$ ($O(n^{2s}h^s|E|)$) time, where k (s) is the Rabin Index (Streett Index) of the game language, and E is the edge relation of the game graph. Besides property synthesis, our results imply improved non-emptiness checks for tree automata, and have applications to decision problems for branching time temporal logics.

The computational complexity of deciding the winner in a Chain game is a significant open problem with tremendous ramifications. In terms of assessing the worst-case strategy size produced by a synthesis procedure, there is a gap between the lower bound and upper bound for Streett games; there is possibly a better synthesis procedure for Streett games. In addition, we have to finalize a proof for the lower bound on strategy size for the general Muller game.

In Chapter 6 we addressed the synthesis of minimum-state strategies for GSP games. We proved that synthesizing the minimum-state memory-based or non-memory based strategy even for the player with the closed winning condition is NP-hard. We introduced subset FSMs as a means of synthesizing minimum-state strategies. Minimum-state memory-based strategy synthesis was cast as a minimum variable Boolean constraints satisfaction problem, as was minimum-state strategy FSM synthesis for the closed winning condition.

Not every synthesis problem presents itself as a two-person GSP game. Chapter 4 examined two-person Gale-Stewart games of incomplete information (GSI game). While we can decide which of the players wins and synthesize the winner's strategy by

defining two GSP games, the complexity and the size of strategies required appears higher, and these games may not have a winner. GSI games enable the modeling of the Watanabe-Brayton [58] component FSM synthesis problem.

It remains to construct instances where the complexity of the larger strategies required in GSI games is exemplified.

Fair games (FGSP game), where each player is required to adhere to his fairness constraint or induce the other player to violate her fairness constraint, were considered in Chapter 5. FGSP games were again decided by defining two GSP games. If either of the player's fairness constraints is a topologically closed set the game has a winner, otherwise the game may not have a winner.

In Chapter 7 we investigated the problem of synthesizing uninitialized strategies—strategy FSMs with no initial state—for GSP games. These games, even closed games, may not have a winner. We presented an algorithm to decide if the player with the closed winning condition has a winning uninitialized strategy; this yields a synthesis algorithm for synthesis for safe-replaceability that can exploit the maximum flexibility available. The general case of non-closed winning conditions remains open, and we conjecture that the problem may be undecidable.

While all the variants of the GSP game may not admit a winner with a deterministic strategy, it would be interesting to consider randomized strategies and see if a player can guarantee a certain probability of winning.

In this thesis, we presented algorithms for deciding the synthesis of properties presented as deterministic ω -automata that was polynomial in the number of states and exponential only in the structural complexity of the property. While combinational logic synthesis has matured into commercialization, sequential synthesis has a long way to go. There are two important issues that will need to be addressed:

1. Binary Decision Diagrams (BDDs) [5] are a canonical data structure for the representation of Boolean functions that has made tremendous advances in the size of FSMs and representable and analyzed. Even though an algorithm's computational complexity may be linear or polynomial in the number of states, it is vital that the algorithm be able to exploit BDDs which enable several states

to be dealt with “atomic” operations. We need to implement our synthesis algorithm with BDDs and assess their performance.

2. Secondly, often properties and components in a FSM network are presented in a multi-level logic form, rather than explicit state transition graphs [56]. This adds another level of complexity to our algorithms that we need to cope with.

We have developed a simple game program with a graphical interface to “play” with games on game graphs. We plan to enhance this progressively and experiment in an effort to push the frontier of practically solvable synthesis problems.

Bibliography

- [1] A. Aziz, F. Balarin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Sequential synthesis using SIS. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 612–617, November 1995.
- [2] F. Balarin and A. L. Sangiovanni-Vincentelli. An iterative approach to language containment. In *Computer Aided Verification 1993*, volume 697 of *LNCS*, pages 29–40. Springer-Verlag, 1993.
- [3] B. Alpern and F. B. Schneider. Recognizing Safety and Liveness. *Distributed Computing*, 2(3):117–126, 1987.
- [4] R. K. Brayton and F. Somenzi. An Exact Minimizer for Boolean Relations. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 316–319, November 1989.
- [5] R. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35:677–691, August 1986.
- [6] J. R. Buchi. On a Decision Method in Restricted Second Order Arithmetic. In *International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11, 1960.
- [7] J. R. Buchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Transaction of the American Mathematical Society*, 138:295–311, 1969.

- [8] O. Carton. Chain Automata. In *IFIP 13th World Computer Congress*, pages 451–458, August 1994.
- [9] A. Church. Logic, arithmetic, and automata. In *Proc. Intern. Congr. Math. 1962*, pages 21–35, 1963.
- [10] O. Coudert. On solving binate covering problems. In *Proceedings of the 33rd Design Automation Conference*, pages 197–202, June 1996.
- [11] M. Davis. Infinite games of perfect information. In *Advances in Game Theory*, pages 85–101. Princeton University Press, 1964.
- [12] S. Dziembowski, M. Jurdzinski, and I. Walukiewicz. How Much Memory is Needed to Win Infinite Games? In *Proc. of the IEEE Symposium on Logic in Computer Science*, pages 99–110, 1997.
- [13] E. A. Emerson. Automata, tableaux, and temporal logics. In *Logics of Programs*, LNCS, pages 79–88. Springer-Verlag, 1985.
- [14] E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. In *Proc. of the Symp. on Foundations of Computer Science*, pages 328–337, October 1988.
- [15] E. A. Emerson and C. S. Jutla. Trees automata, Mu-calculus and determinacy. In *Proc. of the Symp. on Foundations of Computer Science*, pages 368–377, October 1991.
- [16] E. A. Emerson and C. L. Lei. Modalities for Model Checking: Branching Time Logic Strikes Back. *Science of Computer Programming*, 8(3):275–306, June 1987.
- [17] D. Gale and F. M. Stewart. Infinite games with perfect information. In *Contributions to the Theory of Games*, pages 245–266. Princeton University Press, 1953.
- [18] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

- [19] The VIS Group. *VIS: Verification Interacting with Synthesis*, <http://www-cad.eecs.berkeley.edu/Respep/Research/vis>.
- [20] Y. Gurevich and L. Harrington. Trees, automata, and games. In *Proc. of the ACM Symposium on the Theory of Computing*, pages 60–65, May 1982.
- [21] R. Hojati, R. K. Brayton, and R. P. Kurshan. BDD-Based Debugging of Design Using Language Containment and Fair CTL. In C. Courcoubetis, editor, *Proc. of the Conf. on Computer-Aided Verification*, volume 697, pages 41–58. Springer-Verlag, June 1993.
- [22] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, New York, 1979.
- [23] T. Y-K. Kam, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. *Synthesis of Finite State Machines: Functional Optimization*. Kluwer Academic Publishers, 1997.
- [24] M. Kaminski. A Classification of ω -regular languages. *Theoretical Computer Science*, 36:217–229, 1985.
- [25] S.P. Khatri, A. Narayan, S. C. Krishnan, K. L. McMillan, R. K. Brayton, and A. Sangiovanni-vincentelli. Engineering change in a non-deterministic FSM setting. In *Proceedings of the 33rd Design Automation Conference*, pages 451–456, June 1996.
- [26] S. C. Krishnan, A. Puri, and R. K. Brayton. Structural Complexity of ω -automata. In *Symposium on Theoretical Aspects of Computer Science*, volume 900 of *LNCS*, pages 143–156. Springer-Verlag, 1995.
- [27] S. C. Krishnan, A. Puri, R. K. Brayton, and P. P. Varaiya. The Rabin Index and Chain automata, with applications to automata and games. In *Computer Aided Verification 1995*, volume 939 of *LNCS*, pages 253–266. Springer-Verlag, 1995.

- [28] R. P. Kurshan. *Computer-aided Verification of Coordinating Processes: the Automata-theoretic approach*. Princeton University Press, 1994.
- [29] L. H. Landweber. Decision problem for ω -automata. *Mathematical System Theory*, 3:376–384, 1969.
- [30] H. Lescow. On polynomial-size programs winning finite-state games. In *Computer Aided Verification 1995*, volume 939 of *LNCS*, pages 239–252. Springer-Verlag, 1995.
- [31] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer Verlag, 1992.
- [32] D. A. Martin. Borel Determinacy. *Annals of Mathematics*, 102:363–371, 1975.
- [33] Kenneth L. McMillan. Personal communication, August 1994.
- [34] R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65:149–184, 1993.
- [35] M. Michel. Complementation is more difficult with automata on infinite words. Technical report, CNET, Paris, 1988.
- [36] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proc. of the 5th GI conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer Verlag, 1981.
- [37] C. P. Pfleeger. State reduction in incompletely specified finite-state machines. *IEEE Transactions on Computers*, C-22(12), 1973.
- [38] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. of the ACM Symposium on Principles of Programming Languages*, pages 179–180, 1989.
- [39] Anuj Puri. *Theory of hybrid systems and discrete event systems*. PhD thesis, University of California Berkeley, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, December 1995.

- [40] M. O. Rabin. *Automata on Infinite Objects and Church's Problem*, volume 13 of *Regional Conf. Series in Mathematics*. American Mathematical Society, Providence, Rhode Island, 1972.
- [41] M. O. Rabin and D. Scott. Finite Automata and their Decision Problems. *IBM Journal of Research and Development*, 3(2):115–125, 1959.
- [42] S. Safra. Complexity of ω -Automata. In *Proc. of the Symp. on Foundations of Computer Science*, pages 319–327, 1988.
- [43] S. Safra. Exponential Determinization for ω -Automata with Strong-Fairness Acceptance Condition. In *Proc. of the ACM Symposium on the Theory of Computing*, 1992.
- [44] S. Safra and M. Y. Vardi. On ω -Automata and Temporal Logic. In *Proc. of the ACM Symposium on the Theory of Computing*, pages 127–137, May 1989.
- [45] Shmuel Safra. *Complexity of Automata on Infinite Objects*. PhD thesis, The Weizmann Institute of Science, Rehovot, Israel, March 1989.
- [46] V. Singhal and C. Pixley. The Verification Problem for Safe Replaceability. In D. L. Dill, editor, *Proc. of the Conf. on Computer-Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 311–323. Springer-Verlag, June 1994.
- [47] Vigyan Singhal. *Design Replacements for Sequential Circuits*. PhD thesis, University of California Berkeley, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, March 1996.
- [48] A. P. Sistla, M. Y. Vardi, and P. L. Wolper. The Complementation Problem for Büchi Automata, with Applications to Temporal Logic. *Theoretical Computer Science*, 49:217–237, 1987.
- [49] L. Staiger. Finite State ω languages. *Journal of Computer and System Sciences*, 27:434–448, 1983.

- [50] L. Staiger. Research in the theory of ω -languages. *Journal of Information Processing and Cybernetics*, 23:415–439, 1987.
- [51] J. G. Thistle and W. M. Wonham. Control of infinite behavior of finite automata. *SIAM Journal on Control and Optimization*, 32(4):1075–1097, 1994.
- [52] W. Thomas. Automata on Infinite Objects. In J. van Leeuwen, editor, *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, pages 133–191. Elsevier Science, 1990.
- [53] W. Thomas. On the synthesis of strategies in infinite games. In *Symposium on Theoretical Aspects of Computer Science*, volume 900 of *LNCS*, pages 1–13. Springer-Verlag, 1995.
- [54] W. Thomas and H. Lescow. Logical specifications of infinite computations. In *A Decade of Concurrency*, volume 803 of *LNCS*, pages 583–621. Springer-Verlag, 1994.
- [55] K. Wagner. On ω -Regular Sets. *Information and Control*, 43:123–177, 1979.
- [56] H. Y. Wang and Brayton. Multi-level logic optimization of FSM networks. In *Proc. of the Intl. Conf. on Computer-Aided Design*, pages 728–735, November 1995.
- [57] Y. Watanabe. *Logic Optimization of Interacting Components in Synchronous Digital Systems*. PhD thesis, U.C. Berkeley, Electronics Research Laboratory, University of California at Berkeley, April 1994. Tech. Report No. UCB/ERL M94/32.
- [58] Y. Watanabe and R. K. Brayton. The Maximum Set of Permissible Behaviors for FSM Networks. In *Proc. of the Intl. Conf. on Computer-Aided Design*, 1993.
- [59] P. Wolfe. The strict determinateness of certain infinite games. *Pacific Journal of Mathematics*, 5:841–847, 1955.