THEORY OF HYBRID SYSTEMS AND DISCRETE EVENT SYSTEMS

by

Anuj Puri

Memorandum No. UCB/ERL M95/113

19 December 1995

THEORY OF HYBRID SYSTEMS AND DISCRETE EVENT SYSTEMS

by

Anuj Puri

Memorandum No. UCB/ERL M95/113

19 December 1995

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

Theory of Hybrid Systems and Discrete Event Systems

by

Anuj Puri

Doctor of Philosophy in Engineering-Electrical Engineering and Computer Sciences

University of California at Berkeley

Professor Pravin Varaiya, Chair

A continuous system has a continuous state space and an evolution law given by a differential or a difference equation. A discrete event system is modeled by an automaton which changes state in response to events. A hybrid system contains both continuous and discrete event sub-systems. In this thesis we study some theoretical problems in the design and analysis of hybrid systems and discrete event systems.

We first consider the reachability question for a hybrid system — is a target state reachable from an initial state? We show that for hybrid automata with rectangular inclusions, the reachability question can be answered in a finite number of steps. Hybrid systems with more general dynamics can be reduced to hybrid systems with rectangular inclusions using abstractions.

We next consider an Automated Vehicle Highway System (AVHS) design. We consider the safety question: can there be a collision between two vehicles on the AVHS ? We show that the AVHS is safe provided the controllers in the vehicles satisfy a set of constraints. The constraints require the reach set $Reach_f(X_0, t)$ — the set of states reached after time t starting from an initial set X_0 for a differential inclusion $\dot{x} \in f(x)$ — to satisfy a simple criterion. We show that this problem is equivalent to solving an optimal control problem.

We then consider some computational questions for differential inclusions. For a Lipschitz differential inclusion $\dot{x} \in f(x)$, we give a method to compute an arbitrary close approximation of $Reach_f(X_0, t)$. For a differential inclusion $\dot{x} \in f(x)$, and any $\epsilon > 0$, we define a finite *sample graph* A^ϵ. Using graph A^ϵ, we can compute the ϵ-invariant sets of the differential inclusion — the sets that remain invariant under ϵ-perturbations in f.

We also consider some dynamical games played on graphs. The synthesis and the control problem for ω-automata can be formulated as a game between two players. We discuss games on ω-automata and the payoff games. We show that ω-automata games do not necessarily have a value when restricted to positional strategies. We exhibit a bound on the amount of memory required to play these games. We then consider the discounted and mean payoff games. We present the successive approximation and the policy iteration algorithm for solving payoff games. We then show that an ω-automata game with the chain acceptance condition can be solved as a mean payoff game. Solving a chain game is equivalent to solving the model checking problem for propositional μ-calculus. Hence, the policy iteration method can be used to model check μ-calculus formulae. This is at present the most efficient algorithm for model checking propositional μ-calculus.

Professor Pravin Varaiya
Dissertation Committee Chair

To my parents

Acknowledgements

I would like to thank Prof. Pravin Varaiya for his guidance and support, and for allowing me to pursue my ideas. I thank Sriram Krishnan for many interesting discussions on ω-automata and games. I thank Tom Henzinger for an opportunity to collaborate with him. I also thank Prof. Brayton and Prof. Oren for serving on my thesis committee, and Prof. Sastry for serving on my qualifying exam committee. I thank Sonia Sachs, Jennifer McManis, Akash Deshpande and other colleagues for fruitful discussions on hybrid systems and ω-automata, and Wen Hsu, Kumud Sanwal, Karim Toussi, James Yee and Zoran Cvetkovic for discussions on things other than hybrid systems.

Finally, I am most grateful to my parents, Anil and Chi for their love and support during the time I have been at Berkeley.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Continuous systems have been the focus of system theory. But today, due to advances in communications and computer technology, systems are increasingly *event-driven. Discrete event systems* consist of entities or processes which exchange messages and coordinate to perform a task. Examples include computer protocols, and digital controllers in computer systems. When a system includes both discrete event features and continuous activities, we call it a *hybrid system.* An example is the Automated Vehicle Highway System (AVHS) discussed in Chapter 3. In an AVHS, vehicles coordinate by exchanging messages; but a vehicle also controls continuous parameters such as its speed and acceleration.

Continuous, discrete event, and hybrid systems have different modeling paradigms. A continuous system has a continuous state space and an evolution law given by a differential or a difference equation. A discrete event system is modeled by an automaton. An automaton is a graph whose vertices are the states of the system. The edges in the graph are labeled by events. The system jumps from one state to another in response to events or by generating events. A hybrid system combines the two models. The state of the system is the pair (l, x) where l is the vertex in a graph, and x is the continuous state. There is a differential equation at each vertex in the graph. The continuous state x follows the differential equation at the vertex. The system jumps from one vertex to another in response to events or by generating events. The events that get generated may depend on the continuous state.

Systematic methods are needed for design and analysis of hybrid and discrete event systems. Because systems being designed are complex, computer algorithms and computer tools must play a key role in the design process. A *theory* must address the *fundamental* questions faced in the design of hybrid and discrete event systems. What are these fundamental questions ?

The verification question — is the designed system correct — is a fundamental problem. The simplest verification problem is a reachability question — are some "bad" states reachable from the initial state of the system ?

The synthesis problem — synthesize a system to meet a given specification — is another fundamental problem. It is most elegantly formulated as a game between the "controller" who is trying to satisfy the specification, and "disturbance" which wants to violate it.

In this thesis, we address the verification question for hybrid and continuous systems, and the synthesis question for discrete event systems.

In Chapter 2, we study the reachability problem for hybrid systems. The reachability problem asks: is there a trajectory of the hybrid system from an initial state s_0 to a target state? The reachability problem is conceptually straightforward for finite state systems. This is not the case for hybrid systems since the state space is infinite. We show that for initialized hybrid automata with rectangular inclusions, the reachability problem can be solved in a finite number of steps.

In Chapter 3, we study the Automated Vehicle Highway System (AVHS) architecture of Varaiya [41]. We are interested in the safety question: can there be a collision between two vehicles on the AVHS? Using the abstraction methodology in [31], we show that the AVHS is safe provided the controllers in the vehicles satisfy a set of constraints. The problem of checking whether the controllers satisfy the constraints is equivalent to solving an optimal control problem.

In Chapter 4, we consider computational questions about differential inclusions. For a Lipschitz differential inclusion $\dot{x} \in f(x)$, we give a method to compute an arbitrarily close approximation of $Reach_f(X_0, t)$ — the set of states reached after time t starting from an initial set X_0. For a differential inclusion $\dot{x} \in f(x)$, and any $\epsilon > 0$, we define a finite *sample graph* A^ϵ. Using graph A^ϵ, we can compute the

ϵ-*invariant* sets of the differential inclusion — the sets that remain invariant under ϵ-perturbations in f.

In Chapter 5, we study two classes of games played on graphs: games on ω-automata and the payoff games. We show that games on ω-automtata do not necessarily have a value when restricted to positional strategies. We exhibit a bound on the amount of memory required to play such games. We then study the discounted and mean payoff games. We provide a new proof that mean payoff games have optimal positional strategies. We present the successive approximation and the policy iteration algorithm for solving these games. We then show that an ω-automata game with the chain acceptance condition can be solved as a mean payoff game. Solving a chain game is equivalent to solving the model checking problem for propositional μ-calculus. Hence, the policy iteration algorithm can be used to model check μ-calculus formulae. This is at present the most efficient algorithm for model checking propositional μ-calculus.

Some of the work we present in this thesis has also been published in [17, 34, 33, 32].

Chapter 2

Decidable Hybrid Systems

2.1 Introduction

Complex systems that are being designed today incorporate both differential equations to model the continuous behavior and discrete event systems to model instantaneous state changes in response to events. System that incorporate both dynamical and discrete event models are called hybrid systems.

In this chapter we study properties of hybrid automata — a formalism for specifying hybrid systems. In particular, we are interested in algorithmic methods for solving problems relating to hybrid systems. Hybrid systems in which a problem can be solved algorithmically in a finite number of steps are called decidable hybrid systems. We consider the reachability problem: is there a trajectory from an initial state s_0 to a target state. We show that for initialized hybrid automata with constant decoupled differential inclusions, the reachability problem can be solved in a finite number of steps.

In Section 2.2, we discuss transition systems, and the relationships between them. A transition system specifies the states of the system, a set of generators, and the behavior of the system under the generators. In Section 2.3, we introduce hybrid automata and their transition systems. In Section 2.4, we discuss timed automata, a decidable class of hybrid automata. We show that initialized multirate automata are isomorphic to timed automata and are therefore decidable. In Section 2.5, we

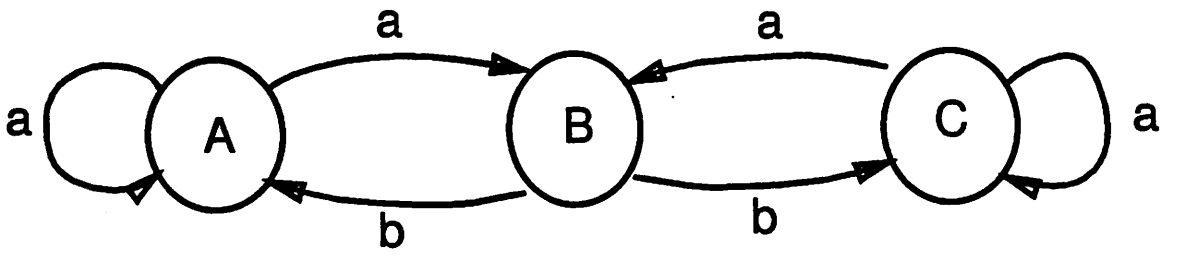

Figure 2.1: An Example Transition System

extend the decidability results to rectangular automata with rectangular differential inclusions.

2.2 State Transition Systems

2.2.1 Transition Systems

Given a system with states S and a set of generators Σ, a *transition system* describes how the generators cause the state to evolve.

Example 2.2.1 *The example of figure 2.1 is a state machine with states* $S = \{A, B, C\}$, *and generators* $\Sigma = \{a, b\}$. *Generator a causes a move from state A to A or B, and a move from state C to C or B. Generator b causes a move from state B to A or C.*

Definition 2.2.1 *A transition system is* $\mathcal{A} = \langle S, \longrightarrow, \Sigma \rangle$ *where*

- S *is a set of states.*
- Σ *is a set of generators.*
- $\longrightarrow \subset S \times \Sigma \times S$ *is the transition relation.*

We will write $(s, \sigma, s') \in \longrightarrow$ as $s \xrightarrow{\sigma} s'$. The transition relation for the system in Example 2.2.1 is $\longrightarrow = \{(A, a, A), (A, a, B), (B, b, A), (B, b, C), (C, a, C), (C, a, B)\}$.

Example 2.2.2 *The definition of a transition system is quite general. A differential equation* $\dot{x} = f(x)$ *with solution* $\phi(t)$ *defines the transition system* $\mathcal{A} = (\mathcal{R}^n, \longrightarrow, Time)$ *where* $Time = \{t | t \in \mathcal{R}^+\}$, *and the transition relation is* $x \xrightarrow{t} y$ *provided* $y = \phi(t)$ *and* $\phi(0) = x$.

Definition 2.2.2 *If* $\mathcal{A} = \langle S, \longrightarrow, \Sigma \rangle$ *is a transition system, then the reversed system is* $\mathcal{A}^{-1} = \langle S, \longrightarrow_R, \Sigma \rangle$ *where* $s' \xrightarrow{\sigma}_R s$ *iff* $s \xrightarrow{\sigma} s'$.

The reversed system moves backwards. For the example of Figure 2.1, the reversed system is the same state machine with edges reversed. In Example 2.2.2, the reversed system $\mathcal{A}^{-1} = (\mathcal{R}^n, \longrightarrow_R, Time)$ is the transition system for the differential equation $\dot{x} = -f(x)$. The reversed system corresponds to the system obtained by reversing the time flow.

2.2.2 Relationship Between Transition Systems

We study relationships between transition systems. For example, two transition systems may be isomorphic. This is a very strong relationship. Weaker relationships are obtained by using the idea of simulation. A transition system simulates another when it can perform the same sequence of actions as the other transition system. Using the idea of simulation, one can define bisimulation between two systems, where each system simulates the other. In our discussion of transition systems $\mathcal{A} = \langle X, \longrightarrow, \Sigma_X \rangle$ and $\mathcal{B} = \langle Y, \longrightarrow, \Sigma_Y \rangle$, we will assume there is a one-to-one correspondence between the generators Σ_X and Σ_Y. For generator $\sigma_x \in \Sigma_X$, we write σ_y for the corresponding generator in Σ_Y.

Definition 2.2.3 *Transition systems* $\mathcal{A} = \langle X, \longrightarrow, \Sigma_X \rangle$ *and* $\mathcal{B} = \langle Y, \longrightarrow, \Sigma_Y \rangle$ *are isomorphic provided there is a bijection* $h : X \longrightarrow Y$ *such that* $x \xrightarrow{\sigma_x} z$ *iff* $h(x) \xrightarrow{\sigma_y} h(z)$.

The transition system of the state machine of Figure 2.1 is isomorphic to the transition system of a state machine obtained from Figure 2.1 by relabeling of the states. A more interesting example is obtained by a change of variables in a differential equation.

Example 2.2.3 *For $x, y \in \mathcal{R}^2$ and differential equation $\dot{x} = (x_1, -x_2)$, consider the change of variables $y = (y_1, y_2) = h(x) = (x_1 - x_2, x_2)$. Then $\dot{y} = g(y) = (y_1 + 2y_2, -y_2)$. The transition system $\mathcal{X} = \langle \mathcal{R}^2, \longrightarrow, Time \rangle$ of differential equation $\dot{x} = f(x)$ and $\mathcal{Y} = \langle \mathcal{R}^2, \longrightarrow, Time \rangle$ of differential equation $\dot{y} = g(y)$ are isomorphic.*

Definition 2.2.4 *Given transition systems $\mathcal{A} = \langle X, \longrightarrow, \Sigma_X \rangle$ and $\mathcal{B} = \langle Y, \longrightarrow, \Sigma_Y \rangle$, we say $\mathcal{B}$ simulates $\mathcal{A}$ with relation $R \subset X \times Y$ if $(x, y) \in R$ and $x \xrightarrow{\sigma_x} x'$ implies that there is $y' \in Y$ such that $y \xrightarrow{\sigma_y} y'$ and $(x', y') \in R$.*

The relation R in Definition 2.2.4 is called a simulation relation for the following reason. Whenever $(x, y) \in R$, for any sequence generated by $\mathcal{A}$ starting from state x, the corresponding sequence can be generated by $\mathcal{B}$ starting from state y. That is, $\mathcal{B}$ from state y can simulate $\mathcal{A}$ from state x when $(x, y) \in R$.

The following lemma states that the simulation property is transitive.

Lemma 2.2.1 *If $\mathcal{A}$ simulates $\mathcal{B}$ and $\mathcal{B}$ simulates $\mathcal{C}$, then $\mathcal{A}$ simulates $\mathcal{C}$.*

When $\mathcal{A}$ simulates $\mathcal{B}$ with relation R and $\mathcal{B}$ simulates $\mathcal{A}$ with R^{-1}, we say $\mathcal{A}$ and $\mathcal{B}$ are bisimilar.

Definition 2.2.5 *If $\mathcal{A} = \langle X, \longrightarrow, \Sigma_X \rangle$ and $\mathcal{B} = \langle Y, \longrightarrow, \Sigma_Y \rangle$ are transition systems, we say $R \subset X \times Y$ is a bisimulation provided $\mathcal{B}$ simulates $\mathcal{A}$ with R and $\mathcal{A}$ simulates $\mathcal{B}$ with $R^{-1} \subset Y \times X$ where $R^{-1} = \{(y, x) | (x, y) \in R\}$.*

A bisimulation between two transition systems indicates that the transition systems are equivalent in some sense. The relation that makes the transition systems bisimilar also indicates the states in the two systems that are equivalent. A stronger equivalence between two transition systems is obtained by requiring that $\mathcal{A}$ and $\mathcal{B}$ are bisimilar, and $\mathcal{A}^{-1}$ and $\mathcal{B}^{-1}$ are bisimilar. That is, the transition systems are bisimilar in reversed time as well.

Definition 2.2.6 *If $\mathcal{A} = \langle X, \longrightarrow, \Sigma_X \rangle$ and $\mathcal{B} = \langle Y, \longrightarrow, \Sigma_Y \rangle$ are transition systems, we say $R \subset X \times Y$ is a time-symmetric bisimulation provided $\mathcal{A}$ and $\mathcal{B}$ are bisimilar and $\mathcal{A}^{-1}$ and $\mathcal{B}^{-1}$ are bisimilar.*

Transition systems that are isomorphic are time-symmetric bisimilar. The next lemma states that the union of bisimulations is itself a bisimulation. This guarantees the existence of a largest bisimulation for two transition systems which are bisimilar.

Lemma 2.2.2 *If* $\mathcal{A} = \langle X, \longrightarrow, \Sigma_X \rangle$ *and* $\mathcal{B} = \langle Y, \longrightarrow, \Sigma_Y \rangle$ *are transition systems with bisimulation* $R_i \subset X \times Y$, $i \in J$, *then* $R = \bigcup_{i \in J} R_i$ *is also a bisimulation.*

2.2.3 Equivalence Relations

Given a transition system $\mathcal{A} = \langle X, \longrightarrow, \Sigma \rangle$, we are interested in identifying states of $\mathcal{A}$ that are "equivalent". We do this by considering the bisimulation of $\mathcal{A}$ with itself. The following lemma shows that a bisimulation on $\mathcal{A}$ leads to an equivalence relation.

Lemma 2.2.3 *If* $\mathcal{A} = \langle X, \longrightarrow, \Sigma \rangle$ *is a transition system and* $R \subset X \times X$ *is a bisimulation on* $\mathcal{A}$ *then the reflexive, symmetric and transitive closure of* R *is also a bisimulation.*

A bisimulation on $\mathcal{A}$ creates an equivalence relation $\sim$ on the states X of $\mathcal{A}$, where all the states in the congruence class $[x]$ are bisimilar to each other. Since the identity relation $\{(x,x) | x \in X\}$ is a bisimulation, by Lemma 2.2.2 there is a largest bisimulation from $\mathcal{A}$ to itself. From Lemma 2.2.3, the largest bisimulation is an equivalence relation.

Definition 2.2.7 *For a transition system* $\mathcal{A} = \langle X, \longrightarrow, \Sigma \rangle$, *and a bisimulation* $\sim \subset X \times X$ *which is an equivalence relation, define the quotient transition system* $\mathcal{A}/\sim = \langle X/\sim, \longrightarrow, \Sigma \rangle$ *where* $X/\sim$ *is the set of congruence classes, and* $[x] \xrightarrow{\sigma} [y]$ *provided* $x \xrightarrow{\sigma} y$.

Although we have stated lemma 2.2.2, lemma 2.2.3 and definition 2.2.7 for bisimulations, they hold for time-symmetric bisimulations as well. For the example in figure 2.1, $R = \{(A,A),(B,B),(C,C),(A,C),(C,A)\}$ is a time-symmetric bisimulation which is an equivalence relation. The quotient transition system is shown in figure 2.2.

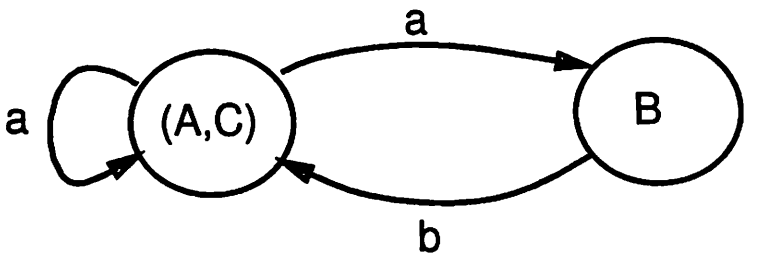

Figure 2.2: Quotient Transition System

2.2.4 Reach Set of Transition Systems

For the transition system $\mathcal{A}= (S, \longrightarrow, \Sigma)$, we define the relation $\Longrightarrow \subset S \times S$, where $\Longrightarrow = \{\xrightarrow{\sigma} | \sigma \in \Sigma\}^*$. That is, $\Longrightarrow$ is the transitive closure of the set $\{\xrightarrow{\sigma} | \sigma \in \Sigma\}$. Essentially $s \Longrightarrow w$ provided w can be reached from s by an application of a sequence of generators from Σ.

Definition 2.2.8 *For a transition system* $\mathcal{A}= \langle S, \longrightarrow, \Sigma\rangle$, *define* $Reach_{\mathcal{A}}(S_0) = \{w | s \Longrightarrow w, s \in S_0\}$.

The reach set $Reach_{\mathcal{A}}(S_0)$ is the set of states that can be reached from S_0 by a sequence of transitions. For a map $h : X \longrightarrow Y$ and $S \subset X$, define $h(S) = \{h(s) | s \in S\}$, and for a relation $R \subset X \times Y$ and $S \subset X$, define $R(S) = \{y | (s, y) \in R$ for some $s \in S\}$.

We next note that when two transition systems are equivalent, the reach set of one can be computed in terms of another.

Lemma 2.2.4 *If* $\mathcal{A} = \langle X, \longrightarrow, \Sigma_X\rangle$ *and* $\mathcal{B} = \langle Y, \longrightarrow, \Sigma_Y\rangle$ *are isomorphic with bijection* $h : X \longrightarrow Y$ *and* $S_0 \subset X$, *then* $Reach_{\mathcal{B}}(h(S_0)) = h(Reach_{\mathcal{A}}(S_0))$.

Lemma 2.2.5 *If* $\mathcal{B} = \langle Y, \longrightarrow, \Sigma_Y\rangle$ *simulates* $\mathcal{A} = \langle X, \longrightarrow, \Sigma_X\rangle$ *with relation* $R \subset X \times Y$ *and* $Y_0 \subset Y$, *then* $Reach_{\mathcal{A}}(R^{-1}(Y_0)) \subset R^{-1}(Reach_{\mathcal{B}}(Y_0))$.

Lemma 2.2.6 *If* $\mathcal{A}^{-1} = \langle X, \longrightarrow_R, \Sigma_X\rangle$ *simulates* $\mathcal{B}^{-1} = \langle Y, \longrightarrow_R, \Sigma_Y\rangle$ *with relation* $R^{-1} \subset Y \times X$ *and* $Y_0 \subset Y$, *then* $R^{-1}(Reach_{\mathcal{B}}(Y_0)) \subset Reach_{\mathcal{A}}(R^{-1}(Y_0))$

As a consequence of lemma 2.2.5 and lemma 2.2.6, we get the following theorem.

Theorem 2.2.1 *If $\mathcal{B}$ simulates $\mathcal{A}$ with $R \subset X \times Y$, $\mathcal{A}^{-1}$ simulates $\mathcal{B}^{-1}$ with R^{-1}, and $Y_0 \subset Y$, then $Reach_{\mathcal{A}}(R^{-1}(Y_0)) = R^{-1}(Reach_{\mathcal{B}}(Y_0))$.*

For more details on transition systems and their relationships, see [27].

2.3 Hybrid Automata

A hybrid automaton [3, 29] models a hybrid system. It consists of control locations with edges between the control locations. The control locations are the vertices in a graph. Each location is labeled with a differential inclusion, and every edge is labeled with a guard, and a jump or reset relation. The state of the hybrid automaton is the pair (l, x) where l is the control location, and $x \in \mathcal{R}^n$ is the continuous state. The hybrid automaton starts from some initial state (l_0, x_0). The trajectory evolves with the control location remaining constant and the continuous state x evolving with the differential inclusion at that location. When the continuous state satisfies the guard of an edge from location l to control location m, a jump can be made to location m. During the jump, the continuous state may get initialized to a new value y. The new state is the pair (m, y). The continuous state y now moves with the new differential inclusion, followed some time later by another jump, and so on.

2.3.1 Syntax

A guard is $g \subset \mathcal{R}^n$. An edge labelled with guard g is enabled when the state $x \in g$. A jump relation is $j \subset \mathcal{R}^n \times \mathcal{R}^n$. During the jump, x is set to y provided $(x, y) \in j$. When j is the identity relation, the continuous state does not change. A differential inclusion is $\dot{x} \in f(x)$ where f is a set-valued map (i.e, $f(x) \subset \mathcal{R}^n$); in case $f(x)$ is a singleton, it is a differential equation. A solution to the differential inclusion with initial condition $x_0 \in \mathcal{R}^n$ is any differentiable function $\phi(t)$, where $\phi : \mathcal{R}^+ \to \mathcal{R}^n$ such that $\phi(0) = x_0$ and $\dot{\phi}(t) \in f(\phi(t))$. Associated with a differential inclusion $\dot{x} \in f(x)$ is the transition system $\langle \mathcal{R}^n, \longrightarrow_f, Time \rangle$ where $x \xrightarrow{t}_f y$ provided for a solution ϕ

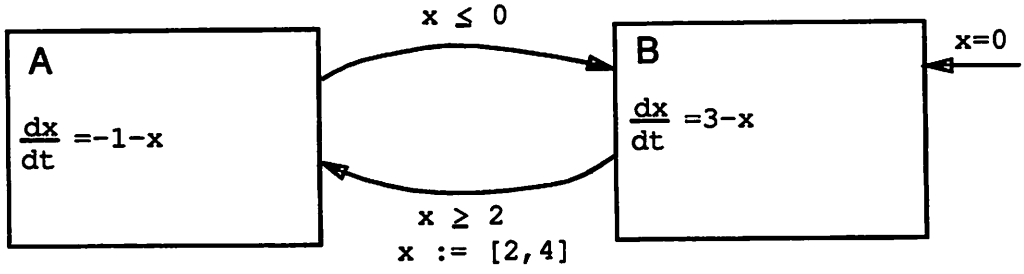

Figure 2.3: Hybrid Automata

of the differential inclusion, $\phi(0) = x$ and $\phi(t) = y$. The set of guards, jump relations and differential inclusions of interest are *Guard*, *Jump*, and *Inclusions* respectively.

A hybrid automaton is $H = (L, D, E)$ where

- L is a set of control locations.
- $D : L \longrightarrow Inclusions$ where $D(l)$ (also written D_l) is the differential inclusion at location l.
- $E \subset L \times Guard \times Jump \times L$ are the edges – an edge $e = (l, g, j, m) \in E$ is an edge from location l to m with guard g and jump relation j.

Figure 2.3 is an example of a hybrid automaton with control locations A and B, and continuous state x. On the edge from B to A, the guard is $g = \{x | x \geq 2\}$, and x is nondeterministically assigned a value in the interval $[2, 4]$.

2.3.2 Transition System of Hybrid Automata

The state space of the hybrid automaton is $Q_H \subset L \times \mathcal{R}^n$. We define the semantics of the hybrid automaton by defining its transition system. The state $(l, x) \in Q_H$ of the hybrid automaton evolves with either the control location remaining fixed and x evolving according to the inclusion at location l; or with a jump from one control location to another. The generators for the hybrid automaton $H = (L, D, E)$ are $\Sigma = Time \cup \{d\}$. The generator $t \in Time$ causes state (l, x) to evolve to state (l, y)

in time t without a change in location. The generator d causes a jump in the control location.

Definition 2.3.1 *For the hybrid automaton $H = (L, D, E)$, with state space $Q_H \subset L \times \mathcal{R}^n$, we define the transition system $H = \langle Q_H, \longrightarrow, Time \cup \{d\} \rangle$ where*

- $(l, x) \xrightarrow{t} (l, y)$ *provided* $x \xrightarrow{t}_{D_l} y$.
- $(l, x) \xrightarrow{d} (m, y)$ *provided* $(l, g, j, m) \in E$, $x \in g$ *and* $(x, y) \in j$.

We next define a τ-transition system for the hybrid automaton in which "*time*" does not play any role.

Definition 2.3.2 *Corresponding to the transition system $H = \langle Q_H, \longrightarrow, Time \cup \{d\} \rangle$ of the hybrid automaton H, define the τ-transition system $H_\tau = \langle Q_H, \longrightarrow, \{\tau\} \cup \{d\} \rangle$ where*

- $(l, x) \xrightarrow{\tau} (l, y)$ *in* H_τ *provided* $(l, x) \xrightarrow{t} (l, y)$ *for some* t *in* H.
- $(l, x) \xrightarrow{d} (m, y)$ *in* H_τ *provided* $(l, x) \xrightarrow{d} (m, y)$ *in* H.

$Reach_H(S_0)$ are all the states that can be reached in the hybrid automaton H starting from $S_0 \subset Q_H$. The next lemma states that the reach set of the transition systems H and H_τ are the same.

Lemma 2.3.1 $Reach_H(S_0) = Reach_{H_\tau}(S_0)$.

2.3.3 Classes of Hybrid Automata

We define some special classes of hybrid automata of interest.

Definition 2.3.3 *A n-dimensional rectangle is a set of the form $r = [l_1, u_1] \times \ldots \times [l_n, u_n]$ with $l_i, u_i \in \mathcal{Z}$. The ith component of r is $r_i = [l_i, u_i]$. The set of all n-dimensional rectangles is $Rect_n$.*

Definition 2.3.4 *A n-dimensional rectangular automaton* $R = (L, D, E)$ *is a hybrid automaton with Inclusions* $= Rect_n$, *Guard* $= Rect_n$ *and Jump* $= \{j | j = j_1 \times \ldots \times j_n$ *where* $j_i = [l_i, u_i]$ *or* $j_i = id\}$, *where relation* $[l_i, u_i] = \{(x, y) | y \in [l_i, u_i]\}$ *and id is the identity relation.*

Definition 2.3.5 *A n-dimensional multirate automaton is a n-dimensional rectangular automaton in which the inclusions consist of single points, i.e., Inclusions* $= \{r : r \in \mathcal{Z}^n\}$.

Figure 2.8 is an example of a rectangular automaton, and figure 2.6 shows a multirate automaton.

Definition 2.3.6 *A n-dimensional timed automaton is a n-dimensional multirate automaton in which the set Inclusions contains the single element* $\{1\}^n$, *i.e.,* $\dot{x}_i = 1$ *for each i at every control location.*

Since the differential equation is fixed at each location in the timed automaton, we denote the timed automaton by $T = (L, E)$.

In an *initialized* rectangular (multirate) automaton, the differential inclusion for the ith component changes only when it is initialized. That is, for locations l and m and edge $(l, g, j, m) \in E$, the inclusion for x_i can be different at l and m provided x_i is initialized during the jump, i.e., j_i is not the identity relation. The multirate automaton in figure 2.6 is an initialized multirate automaton. For example, $\dot{x} = 2$ in location A and $\dot{x} = -3$ in location B, but x is initialized on the jump from A to B.

The ith component of guard g is $g_i = [l_i, u_i]$. For $k > 0$, define $\frac{g_i}{k} = [\frac{l_i}{k}, \frac{u_i}{k}]$, and for $k < 0$, define $\frac{g_i}{k} = [\frac{u_i}{k}, \frac{l_i}{k}]$. Similarly, the ith component of relation j is j_i. For relation $j_i = [l_i, u_i]$, $\frac{j_i}{k}$ is similarly defined, and for $j_i = id$, $\frac{j_i}{k} = id$.

2.3.4 Additional Notation

For an edge $e = (l, j, g, m)$ of a n-dimensional rectangular automaton with $j_i = [r_i, s_i]$ and $g_i = [a_i, b_i]$, we say the edge is labeled with the initialization $(x_i := j_i = [r_i, s_i])$ and the guard $(a_i \leq x_i \leq b_i)$. This information is summarized by writing a

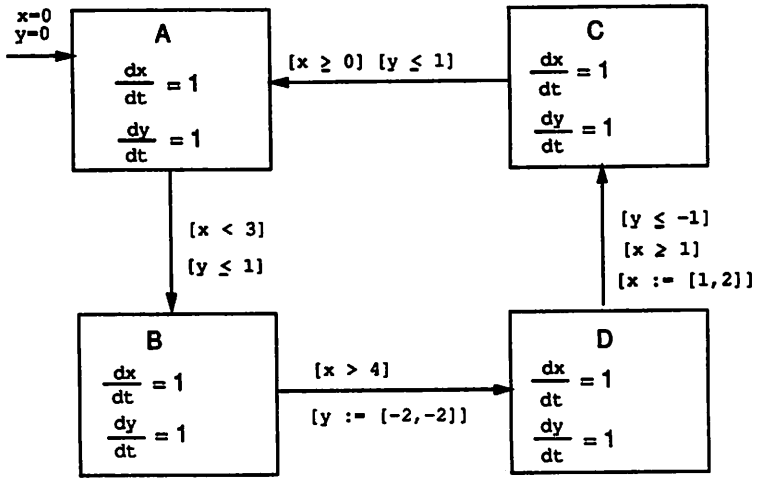

Figure 2.4: Timed Automaton

conjunctive expression $\bigwedge_{i=1}^{n}(a_i \leq x_i \leq b_i)(x_i := j_i)$. We also permit a disjunction of conjunctive expressions of the form $\bigvee_{k=1}^{m}\bigwedge_{i=1}^{n}(a_{ik} \leq x_i \leq b_{ik})(x_i := j_{ik})$. Such an expression would be represented by m edges in our definition of a rectangular automaton.

We also permit expressions of the form $((x_1 \leq 5) \longrightarrow (x_1 := 5))$, which translates to $(x_1 > 5) \vee ((x_1 \leq 5) \wedge (x_1 := 5))$, and would be represented with two edges. In general, any boolean expression can be translated into disjunction of conjunctive expressions, and represented with multiple edges.

2.4 Timed Automata and Initialized Multirate Automata

2.4.1 Timed Automata

A n-dimensional timed automaton has n clocks, $x_i, i = 1, \ldots, n$, with $\dot{x}_i = 1$ at every control location. The guards on the edges are rectangles. The jump relation either leaves the value of the clock unchanged, or sets the clock nondeterministically to a value in an interval. Figure 2.4 is an example of a timed automaton.

We review the work in [4] in which it was shown that timed automata have a time-symmetric bisimulation with a finite number of congruence classes.

Consider a timed automaton $T = (L, E)$ with state space Q_T. Suppose M_i is the largest integer with which x_i is compared in a guard, or assigned in a jump relation. We will define an equivalence relation $\sim$ on $\mathcal{R}^n$, where two states will be related provided the ordering of the fractional parts of the components of the two states is the same; and the integer parts of the components of two states match, or are greater than the largest integer with which they are compared. For $z \in \mathcal{R}$, we write $\lfloor z \rfloor$ for its integer part, and $\langle z \rangle$ for its fractional part.

Definition 2.4.1 *We define a relation* $\sim \subset \mathcal{R}^n \times \mathcal{R}^n$, *where* $x \sim y$ *provided for every* i *and* j

- $\langle x_i \rangle < \langle x_j \rangle$ *iff* $\langle y_i \rangle < \langle y_j \rangle$
- $\langle x_i \rangle = \langle x_j \rangle$ *iff* $\langle y_i \rangle = \langle y_j \rangle$
- $(\lfloor x_i \rfloor = \lfloor y_i \rfloor \leq M_i)$ *or* $((\lfloor x_i \rfloor > M_i)$ *and* $(\lfloor y_i \rfloor > M_i))$

It can be checked that $\sim$ is an equivalence relation.

We next describe the congruence classes of the equivalence relation $\sim$. For a vector $x \in \mathcal{R}^n$, we define $\langle x \rangle$ to be the ordering of the fractional parts of its components. For example, for $x = [0.3\ 4.7\ 3.2\ 8.3]$, $\langle x \rangle = (0 < \langle x_3 \rangle < \langle x_1 \rangle = \langle x_4 \rangle < \langle x_2 \rangle)$. Similarly, $\lfloor x \rfloor$ is a vector of integer parts of the components, or an indication that the integer part is greater than M_i. For $x = [0.3\ 4.7\ 3.2\ 8.3]$, and $M_1 = M_2 = M_3 = M_4 = 5$, $\lfloor x \rfloor = [0\ 4\ 3\ >]$.

Lemma 2.4.1 *For the equivalence relation* $\sim$, *the congruence class* $[x] = \{y | \langle x \rangle = \langle y \rangle$ *and* $\lfloor x \rfloor = \lfloor y \rfloor\}$.

Since the integers M_i are bounded a priori, the equivalence relation $\sim$ has only a finite number of congruence classes. Figure 2.5 shows the congruence classes for $\mathcal{R}^2$. We extend the relation $\sim$ to $Q_T \subset L \times \mathcal{R}^n$ by defining $(l, x) \sim (l, y)$ provided $x \sim y$. The relation $\sim$ is an equivalence relation on Q_T and the congruence class $[(l, x)] = \{l\} \times [x]$. Since L is finite, Q_T has a finite number of congruence classes.

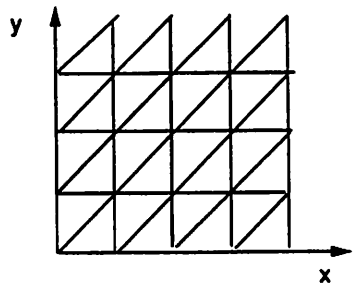

Figure 2.5: Congruence Classes of the Bisimulation for Timed System

Theorem 2.4.1 *The equivalence relation $\sim$ is a time-symmetric bisimulation for the τ-transition system $T_\tau = (Q_T, \longrightarrow, \{\tau\} \cup \{d\})$.*

Therefore the quotient transition system $T_\tau / \sim = (Q_T / \sim, \longrightarrow, \{\tau\} \cup \{d\})$ has only a finite number of states. As a consequence, we get the following result.

Lemma 2.4.2 *For a timed automaton $T = (L, E)$, $Reach_T([S_0]) = Reach_{T_\tau/\sim}([S_0])$ can be computed in a finite number of steps.*

Although, we have only considered timed automata in which the coefficients in the guards and jump relation are integers, similar results hold for timed automata with rational coefficients.

2.4.2 Initialized Multirate Automata

In a multirate automaton, different components of state x move at different rates. In an initialized multirate automaton, the rate at which x_i moves can change when a jump is made into another control location and x_i is initialized during the jump. We show how to construct a timed automaton such that the transition systems of the timed automaton and the initialized multirate automaton are isomorphic.

Corresponding to the initialized n-dimensional multirate automaton $M = (L, D, E)$, define the corresponding n-dimensional timed automaton $T = (L, E_T)$. The edge $(l, g, j, m) \in E$ is replaced with the edge $(l, g^T, j^T, m) \in E_T$ with $g_i^T = \frac{g_i}{v_i}$ and $j_i^T = \frac{j_i}{w_i}$ where $D_l = v$ and $D_m = w$.

Figure 2.6 shows an initialized multirate automaton, and Figure 2.7 shows the corresponding timed automaton. Consider the edge from location A to location B. The guard $(x \leq 5)$ on the edge in the multirate automaton is replaced with the guard $(x \leq \frac{5}{2})$ in the timed automaton since $\dot{x} = 2$ in location A of the multirate automaton. Similarly the initialization $x := [4, 4]$ in the multirate automaton is replaced with the initialization $x := [\frac{4}{-3}, \frac{4}{-3}]$ in the timed automaton since $\dot{x} = -3$ in location B of the multirate automaton.

Define the map $h : L \times \mathcal{R}^n \longrightarrow L \times \mathcal{R}^n$ by $h(l, y) = (l, x)$ where $x = (\frac{y_1}{v_1}, \ldots, \frac{y_n}{v_n})$ and $D_l = v$. The state space of the timed automaton T is $Q_T = h(Q_M)$. The next theorem states that the transition systems of the multirate automaton and the timed automaton are isomorphic.

Theorem 2.4.2 *The transition system $M = (Q_M, \longrightarrow, Time \cup \{d\})$ of the initialized multirate automaton, and $T = (Q_T, \longrightarrow, Time \cup \{d\})$ of the timed automaton are isomorphic with bijection $h : Q_M \longrightarrow Q_T$.*

Lemma 2.4.3 $h(Reach_M(X_0)) = Reach_T(h(X_0))$.

Since the reach set of a timed automaton can be computed in a finite number of steps, the reach set of an initialized multirate automaton can also be computed in a finite number of steps.

2.5 Initialized Rectangular Automata

In this section, we discuss initialized rectangular automata. Decidability of initialized rectangular automata was shown in [30]. We follow the proof given in [17].

In a rectangular automaton, the inclusion for the ith component x_i is $\dot{x}_i \in [l_i, u_i]$. Figure 2.8 shows an initialized rectangular automaton. In this section we present a method to translate an initialized rectangular automaton into an initialized multirate automaton. Given an n-dimensional rectangular automaton R, we define a simulation relation S and a $2n$-dimensional multirate automaton M such that M simulates R, and R^{-1} simulates M^{-1}.

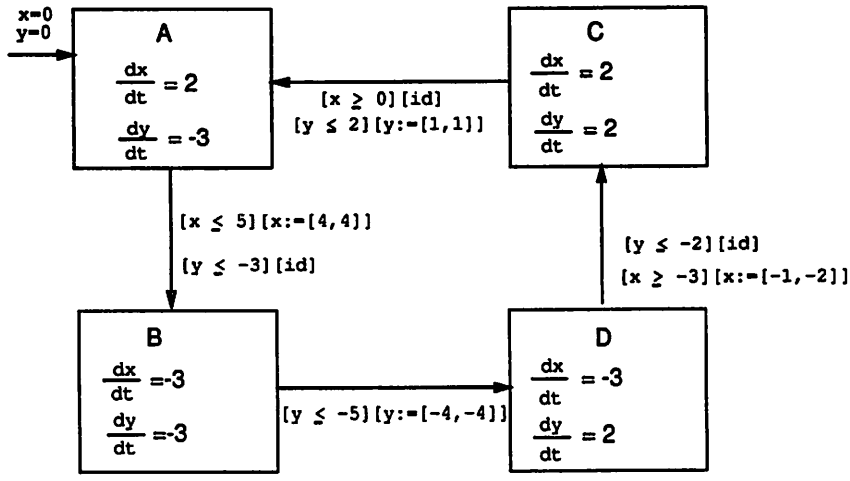

Figure 2.6: Multirate Automaton

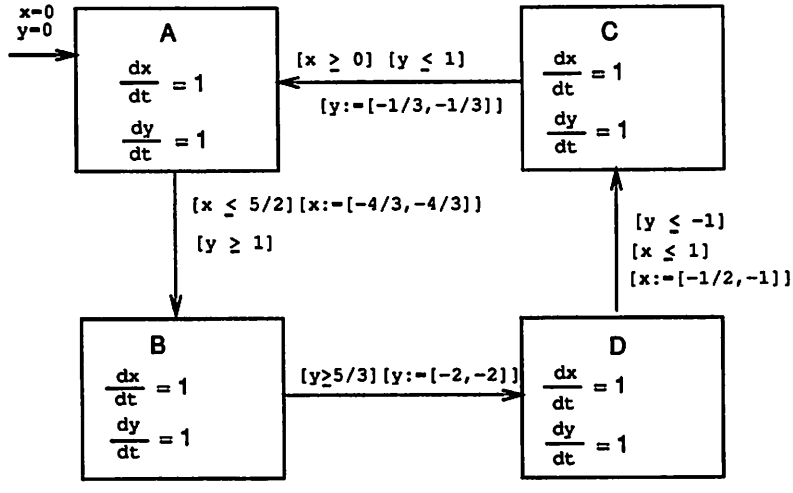

Figure 2.7: Translation of Multirate Automaton to Timed Automaton

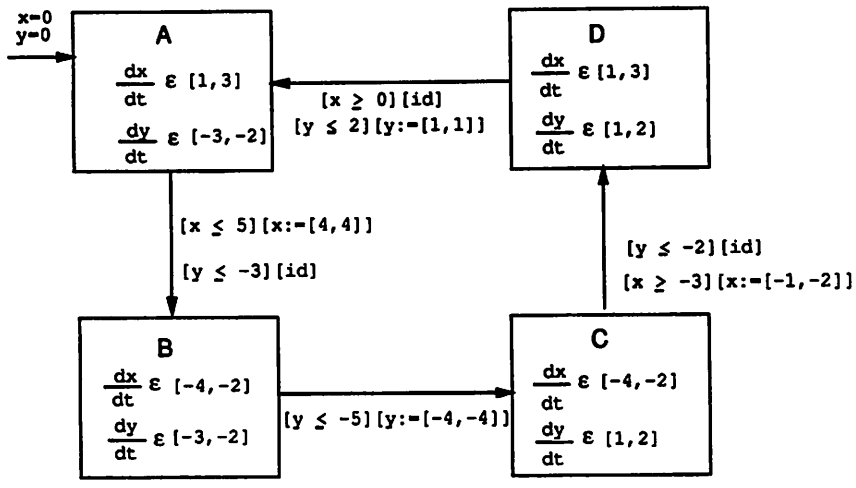

Figure 2.8: Initialized Rectangular Automaton

We do this by replacing a variable $\dot{x} \in [l, u]$ in the rectangular automaton with two variables $\dot{x}_l = l$ and $\dot{x}_u = u$ in the multirate automaton. The variable x defines an envelope in the rectangular automaton whose lower and upper boundaries are tracked by x_l and x_u in the multirate automaton (figure 2.9). When the test $(x \geq a)$ is made in the rectangular automaton, we make the test $(x_u \geq a)$ in the multirate automaton. After the test, the boundary of the envelope gets redefined (figure 2.9). This is done by checking whether $(x_l \leq a)$, and initializing x_l to a when this is the case. Hence, the lower and the upper boundary of the envelope are tracked again by x_l and x_u after the test. Similarly, when the test $(x \leq a)$ is made in the rectangular automaton, we test whether $(x_l \leq a)$ in the multirate automaton. To update the boundary, we initialize x_u to a when $(x_u \geq a)$.

More formally, corresponding to an n-dimensional rectangular automaton $R = (L, D_R, E_R)$, we define the $2n$-dimensional multirate automaton $M = (L, D_M, E_M)$. The variable y_{2i-1} in the multirate automaton tracks the lower boundary of the envelope created by x_i in rectangular automaton, and the variable y_{2i} tracks the upper boundary of the envelope. At location l, $\dot{x}_i \in [l_i, u_i]$ in the rectangular automaton is replaced with $\dot{y}_{2i-1} = l_i$ and $\dot{y}_{2i} = u_i$ in the multirate automaton. A guard $(x_i \geq a)$ in the rectangular automaton is replaced with $(y_{2i} \geq a) \wedge ((y_{2i-1} \leq a) \longrightarrow (y_{2i-1} := a))$ on the corresponding edge in the multirate automaton. Similarly the guard $(x_i \leq a)$

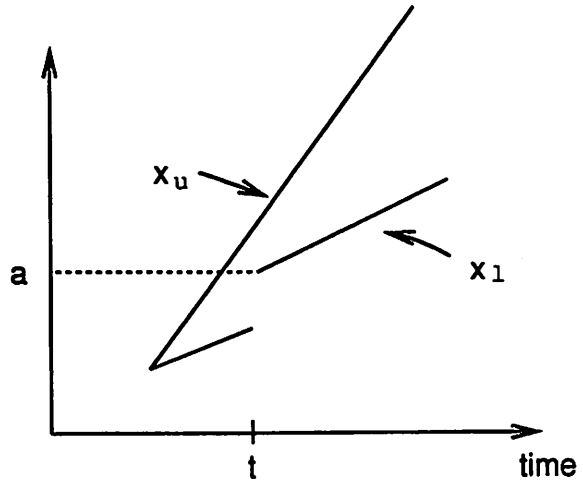

Figure 2.9: Envelope of differential inclusion $\dot{x} \in [l, u]$

in the rectangular automaton is replaced with $(y_{2i-1} \leq a) \wedge ((y_{2i} \geq a) \longrightarrow (y_{2i} := a))$ in the multirate automaton. The initialization $(x_i := [r, s])$ in the rectangular automaton is replaced with $(y_{2i-1} := r) \wedge (y_{2i} := s)$ on the corresponding edge in the multirate automaton. Figure 2.10 is the initialized multirate automaton obtained by translating the initialized rectangular automaton of figure 2.8.

Definition 2.5.1 *For a rectangular automaton R with state space Q_R, and the corresponding multirate automaton M with state space Q_M, define the relation $S \subset Q_M \times Q_R$ where $S = \{((l, y), (l, x)) | y_{2i-1} \leq x_i \leq y_{2i}\}$.*

The relationship between the states of the initialized rectangular automaton, and the initialized multirate automaton obtained by translation from it, is made with the relation S. The multirate automaton tracks the rectangular automaton. When the multirate automaton reaches state (l, y), the rectangular automaton can reach any state in the set $S(\{(l, y)\}) = \{l\} \times [y_1, y_2] \times \cdots \times [y_{2n-1}, y_{2n}]$.

Theorem 2.5.1 *If $R = (Q, \longrightarrow, Time \cup \{d\})$ is the transition system of an initialized rectangular automaton, and $M = (Q_M, \longrightarrow, Time \cup \{d\})$ is the transition system of the corresponding multirate automaton, then*

- *M simulates R with relation S^{-1}, and*

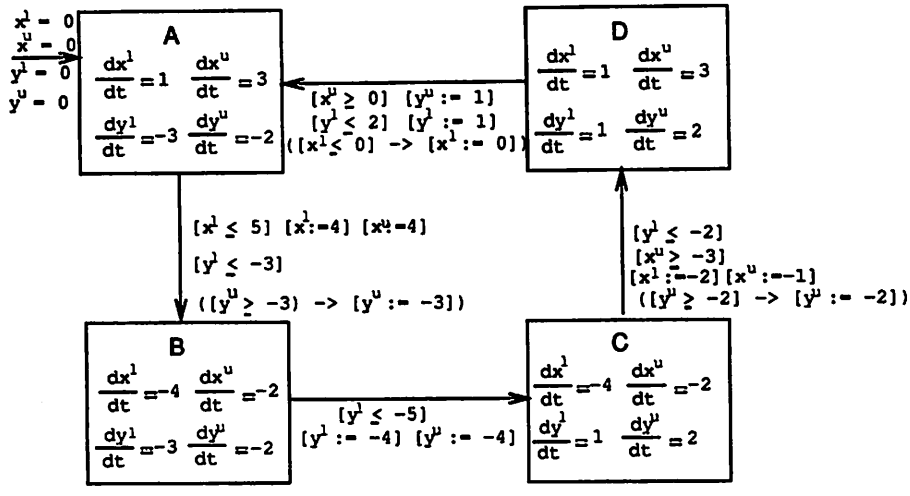

Figure 2.10: Translation of Rectangular Automaton to Multirate Automaton

- R^{-1} *simulates* M^{-1} *with relation* S.

Proof: We will only show that M simulates R with relation S^{-1}. The proof for showing R^{-1} simulates M^{-1} is similar. Suppose $((l,x),(l,y)) \in S^{-1}$, $(l,x) \xrightarrow{t} (l,x')$ and $\dot{x}_i \in [l_i, u_i]$. Since $y_{2i-1} \leq x_i \leq y_{2i}$, we get

$$y'_{2i-1} = y_{2i-1} + l_i t \leq x_i + l_i t \leq x'_i \leq x_i + u_i t \leq y_{2i} + u_i t = y'_{2i}.$$

Therefore $(l,y) \xrightarrow{t} (l,y')$ and $((l,x'),(l,y')) \in S^{-1}$.

Next suppose $((l,x),(l,y)) \in S^{-1}$, $(l,x) \xrightarrow{d} (l,x')$ and $(l,y) \xrightarrow{d} (l,y')$. First consider the case when the ith component of the jump relation is $j_i = [r_i, s_i]$. Then

$$y'_{2i-1} = r_i \leq x'_i \leq s_i = y'_{2i}.$$

Next suppose $j_i = id$, and the ith component of the guard is $g_i = [a_i, b_i]$. Then

$$y'_{2i-1} = max(a_i, y_{2i-1}) \leq x'_i = x_i \leq min(b_i, y_{2i}) = y'_{2i}.$$

Thus $((l,x'),(l,y')) \in S^{-1}$. ∎

From Theorem 2.5.1 and Theorem 2.2.1, we get the following result.

Theorem 2.5.2 $Reach_R(S(Y_0)) = S(Reach_M(Y_0))$.

Since the reach set of an initialized multirate automaton can be computed in a finite number of steps, we can compute the reach set of an initialized rectangular automaton in a finite number of steps.

2.6 Conclusion

In this chapter we have presented a decidable class of hybrid systems which can be analyzed algorithmically. To analyze systems with more complicated dynamics, we use abstractions [31]. A system which has complicated continuous dynamics is abstracted with one which has simpler dynamics, such as rectangular inclusions. The abstractions are conservative approximations. Hence, any property we prove for the abstraction always holds for the original system.

Chapter 3

Driving Safely in Smart Cars

3.1 Introduction

Automation of driving functions is central to proposals for the design of an Automated Vehicle/Highway System (AVHS). In the control architecture in [41], it is proposed that vehicles travel in platoons. Three maneuvers are needed: in *merge*, two platoons join together; in *split*, one platoon separates into two; and in *change lane*, a single vehicle changes lane. Using these maneuvers, a vehicle enters the system, becomes part of a platoon, travels to its destination, detaches itself from the rest of the platoon, and exits out of the system.

The overall architecture is divided into five layers: network layer, link layer, coordination layer, regulation layer and physical layer. The physical layer describes the vehicle dynamics. The regulation layer comprises the set of control laws for acceleration, braking and steering. The control law that is applied depends upon whether the vehicle is a leader or a follower, and upon the commands from the coordination layer (Leader is the lead vehicle of the platoon; other vehicles are called followers). The coordination layer contains protocols. These protocols exchange coordination messages with the other vehicles in order to determine which of three maneuvers to execute and when to do so. The link layer manages a section of the highway, setting the recommended velocity and platoon size for vehicles in that section of the highway. The network layer determines the route for the vehicles.

Designs for the various layers of AVHS have been proposed. A design for the control laws in the regulation layer is proposed in [15] and [14]. In [15], control laws are proposed for the *leader* mode in which a platoon tracks the recommended velocity, or if there is a platoon in front, then it remains a safe distance behind that platoon. Control laws for the *merge* and *split* maneuver are proposed in [14]. A design for the coordination layer is proposed in [21]. This consists of protocols modeled with finite state machines.

In this chapter we consider the following problem. Consider an AVHS, for example the system proposed in [41, 15, 14, 21]. How do we know that such a system is safe? Of course we have to define what safety means. We say a system is *unsafe* if there is a possibility of a high relative velocity collision on the AVHS. We want to *prove* for a proposed design of an AVHS, that there is no such possibility. We can simulate the system. But that only checks safety for a finite number of simulation paths or trajectories of the system. We want to prove safety for every trajectory of the system. In this chapter we develop an approach for proving that a system is safe. We consider a proposed design for an AVHS and show that if the physical controllers in the vehicles satisfy a set of constraints then the AVHS is safe.

In Section 3.2, we describe relevant parts of the AVHS design proposed by [41, 15, 14, 21]. In Section 3.3, we show that a single lane AVHS is safe when the controllers satisfy a set of constraints. In Section 3.4, we extend the design to include the change lane maneuver, and prove that the new design is also safe. In Section 3.5, we conclude with some open problems.

3.2 Single Lane AVHS

In this section we describe the merge and split maneuvers, and the relevant aspects of the coordination and the regulation layer that are important from a safety viewpoint.

3.2.1 Maneuvers and Architecture

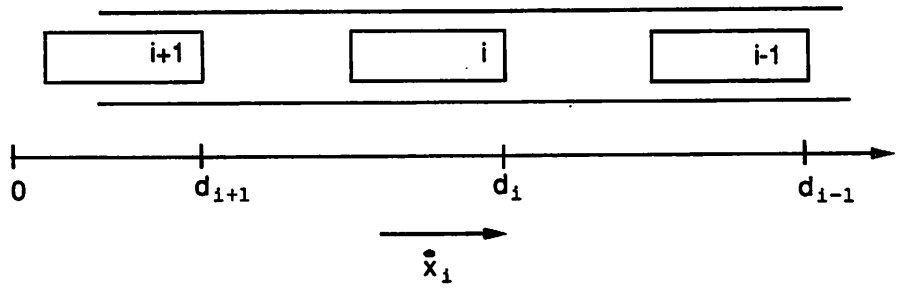

Figure 3.1: A Lane of the Highway

Figure 3.1 shows a lane of the highway with different platoons. The vehicle at the head of the platoon is called the leader, and it is normally under the *leader* control law. Under the leader control law, it follows the platoon in front at a safe distance and speed. The other vehicles in the platoon are under the *follow* control which tracks the leader of the platoon. At certain times, the leader of the platoon may decide to merge with the platoon in front of it. To do this, it communicates with the platoon in front, and if permitted, it orders the regulation layer to follow the *merge* control. This causes it to merge with the platoon in front. On the other hand, if during the merge maneuver, the platoon in front suddenly decelerates or behaves erratically, the merge maneuver is aborted and a switch is made to the *abort* control. The *abort* control law steers the leader to a state from which it is safe to switch back to *leader* control. When the merge is successful, the platoon which was merging becomes part of the platoon in front, and the leader switches to the *follow* control. Similarly, a vehicle in a platoon may decide that it wants to split from the platoon. In this case, it communicates with the leader of the platoon, and if the leader permits, the vehicle orders its regulation layer to follow the *split* control law, which causes the vehicle and all vehicles behind it to split and become a separate platoon.

As shown in figure 3.1, the distance of platoon i from the origin is d_i. The continuous state of the leader of platoon i is x_i. In [15], it is assumed $x_i = (d_i, \dot{d}_i, \ddot{d}_i)$.

The Regulation Layer has five control laws: leader ($\dot{x}_i = L(x_i, x_{i-1})$), merge ($\dot{x}_i = M(x_i, x_{i-1})$), follow, split ($\dot{x}_i = S(x_i, x_{i-1})$), and abort ($\dot{x}_i = A(x_i, x_{i-1})$). The regulation layer of a vehicle is in the *follow* mode when the vehicle is not the leader

of the platoon. We assume that a vehicle in the *follow* mode exactly tracks the leader of the platoon. Hence, the continuous state of platoon i is x_i, the same as its leader's state. The continuous state of the AVHS is $x = (x_0, x_1, x_2, \ldots)$. Notice that the control for a platoon depends on its own state and the state of the platoon in front.

3.2.2 Safety Criterion and Control Table

In a lane of the highway, the vehicles will be going through a sequence of modes such as *merge, split, follow, abort,* or *leader.* We want to prove that at no point in time is there a high relative velocity collision between any two vehicles. A high relative velocity collision is defined as a collision in which $\dot{d}_i - \dot{d}_{i-1} \geq c$ m/s, where c is a design parameter.

Associated with each control law f (where f could be the *leader, merge* or the *split* control) are two sets: an initial set $\mathcal{S}_f$ and an unsafe set U_f. The control f starts from an initial condition $(x_i(0), x_{i-1}(0)) \in \mathcal{S}_f$. The unsafe set U_f is the set of undesirable states which should be unreachable; for example, U_f is the set of states representing collision between vehicles. The initial set $\mathcal{S}_f$ is chosen so that starting from an initial condition $(x_i(0), x_{i-1}(0)) \in \mathcal{S}_f$, the unsafe set U_f is unreachable. Before the control $\dot{x}_i = f(x_i, x_{i-1})$ is applied, it is checked that the initial state $(x_i, x_{i-1}) \in \mathcal{S}_f$.

The set U_f can depend on control f. For the merge control, we require that there be no high relative velocity collision; for the leader control, we impose the stronger condition that there be no collision. We summarize this information in the *Control Table* (table 3.1). The control table shows when a particular control is permissible. The control law $\dot{x}_i = f(x_i, x_{i-1})$ can be applied provided $(x_i, x_{i-1}) \in \mathcal{S}_f$. Before a control is applied, a check is made in the Control Table to see whether the initial state $(x_i, x_{i-1}) \in \mathcal{S}_f$.

In the abort control law, a vehicle applies full brakes. Therefore, it is safe to switch to the abort control law at any time from any of the other controls.

Control Table			
	Control	Initial Set	Unsafe Set
Leader	$\dot{x}_i = L(x_i, x_{i-1})$	$\mathcal{S}_L$	$U_L = \{x \mid d_i = d_{i-1}\}$
Merge:	$\dot{x}_i = M(x_i, x_{i-1})$	$\mathcal{S}_M$	$U_M = \{x \mid d_i = d_{i-1}$ and $\dot{d}_i - \dot{d}_{i-1} \geq c\}$
Split:	$\dot{x}_i = S(x_i, x_{i-1})$	$\mathcal{S}_S$	U_S
Abort:	$\dot{x}_i = A(x_i, x_{i-1})$		

Table 3.1: Control Table to Check Safety

3.3 Safe Driving, Abstractions and Optimal Control

Consider the single lane system of figure 3.1. Each vehicle follows the control

$$\dot{x}_i = f(x_i, x_{i-1}), \tag{3.1}$$

where $(x_i(0), x_{i-1}(0)) \in \mathcal{S}_f$. The control f could be the leader, merge, split, or the abort control law. We must show that the unsafe set U_f is unreachable in each case.

We face the problem that the control for vehicle $i-1$ depends on vehicle $i-2$, which depends on vehicle $i-3$, and so on. To avoid working with an infinite dimensional system, we use a conservative abstraction. We look at the dynamics between two vehicles, vehicle i and vehicle $i-1$, and abstract the differential equation for vehicle $i-1$,

$$\dot{x}_{i-1} = f(x_{i-1}, x_{i-2}), \tag{3.2}$$

with the differential inclusion

$$\ddot{d}_{i-1} \in [A_{min}, A_{max}]. \tag{3.3}$$

We choose A_{min} to be the maximum deceleration (full brakes), and A_{max} to be the maximum acceleration (full throttle). This implies that for any law f in equation 3.2, the solution for equation 3.2 is contained in the set of solutions for equation 3.3. In this sense, equation 3.3 is a (conservative) abstraction of equation 3.2.

We now prove safety for the abstracted system

$$\dot{x}_i = f(x_i, x_{i-1}), \tag{3.4}$$

$$\ddot{d}_{i-1} \in [A_{min}, A_{max}],$$

$$(x_i(0), x_{i-1}(0)) \in \mathcal{S}_f,$$

by showing that U_f is unreachable. This will imply that the system of equation 3.1 is safe, since the reachable set of equation 3.4 is larger than the reachable set of equation 3.1. Notice that proving safety for equation 3.4 is equivalent to proving that no matter what is the behavior on part of vehicle $i-1$, the control law for vehicle i prevents it from having a high speed collision with vehicle $i-1$. Furthermore, equation 3.4 is independent of vehicles $i-2$ and beyond.

We show that the system defined by equation 3.4 is safe for each control f and initial set $\mathcal{S}_f$. From this it follows that there can be no high relative velocity collision involving vehicle i and vehicle $i-1$. Since i is arbitrary, it follows that for *every* i, there is no high relative velocity collision involving vehicle i and vehicle $i-1$.[1] That is, the system of figure 3.1 is safe when the initial state is such that $(x_{i-1}(0), x_i(0)) \in \mathcal{S}_f$ for each i.

To show that U_f is unreachable in equation 3.4, we need to compute the reach set $Reach_f(\mathcal{S}_f)$ (i.e., all states reachable from $\mathcal{S}_f$ under the law f), and check whether $Reach_f(\mathcal{S}_f) \bigcap U_f = \emptyset$. For a control f, there is also a largest set of safe initial states $\mathcal{S}_f^* = (Reach\_{}_f(U_f))^c$ (i.e., complement of all states which can reach U_f). It is clear that for a set $\mathcal{S}_f$, $Reach_f(\mathcal{S}_f)$ and $\mathcal{S}_f^*$ are invariant sets, and $\mathcal{S}_f \subset Reach(\mathcal{S}_f) \subset \mathcal{S}_f^*$. Instead of explicitly computing the Reach set, we turn our problem into an equivalent optimal control problem.

3.3.1 Optimal Control Problem

To determine whether $U_f = \{x | g(x) \leq 0\}$ is unreachable from $\mathcal{S}_f$ in equation 3.4, we solve the following optimal control problem, with control $u = \ddot{d}_{i-1}$:

$$\begin{aligned} \textit{Cost}: \quad & J = \min_t g(x(t)), \\ \textit{Differential equation}: \quad & \dot{x}_i = f(x_i, x_{i-1}), \\ \textit{Initial condition constraint}: \quad & (x_i(0), x_{i-1}(0)) \in \mathcal{S}_f, \end{aligned} \tag{3.5}$$

[1] A vehicle that is not engaged in the merge maneuver should have no collision.

$$\textit{State constraint}: \quad \dot{d}_i \geq 0,\ \dot{d}_{i-1} \geq 0,$$
$$\textit{Control constraint}: \quad u = \ddot{d}_{i-1} \in [A_{min}, A_{max}].$$

The optimal control finds the choice of the initial condition $(x_i(0), x_{i-1}(0)) \in \mathcal{S}_f$ and control which minimize the cost while remaining within the constraints. Notice that the state constraints require the velocities of both vehicles to be positive. If the optimal cost $J > 0$, then we know that for every initial condition $(x_i, x_{i-1}) \in \mathcal{S}_f$, the set U_f is unreachable in equation 3.4. On the other hand, if $J \leq 0$, then the trajectory which minimizes J also takes equation 3.4 into U_f. Therefore the system of equation 3.4 is safe if and only if the optimal cost $J > 0$ in equation 3.5.

3.3.2 A Leader Control Example

Equation 3.6 shows part of the leader control developed in [15]. The control is applied during safety-critical situations when the inter-vehicle distance is small, or the relative velocity between vehicles is large.

$$\dddot{d}_i = -3\ddot{d}_i - 3(\dot{d}_i - \dot{d}_{i-1}) + ((d_{i-1} - d_i) - (\dot{d}_i + 10)) \tag{3.6}$$

The state of the system is $x = ((d_{i-1} - d_i), \dot{d}_i, \dot{d}_{i-1}, \ddot{d}_i)$. The maximum braking capacity of a vehicle is $A_{min} = -5\frac{m}{s^2}$, and the maximum acceleration is $A_{max} = 2\frac{m}{s^2}$. We choose the initial set $\mathcal{S}_L$ where

$$\mathcal{S}_L = \{(d_{i-1} - d_i) + \frac{(\dot{d}_i^2 - \dot{d}_{i-1}^2)}{2A_{Min}} - 10 - (\dot{d}_i - \dot{d}_{i-1}) \geq 0,$$
$$d_{i-1} - d_i \geq 5, \quad -5 \leq \ddot{d}_i \leq 2,$$
$$0 \leq \dot{d}_i \leq 30, \quad 0 \leq \dot{d}_{i-1} \leq 30\}.$$

We want to determine if a collision between vehicle i and vehicle $i-1$ is possible when vehicle i starts from an initial condition $x(0) \in \mathcal{S}_L$ and follows the control in equation 3.6. To determine this, we solve the following equivalent optimal control

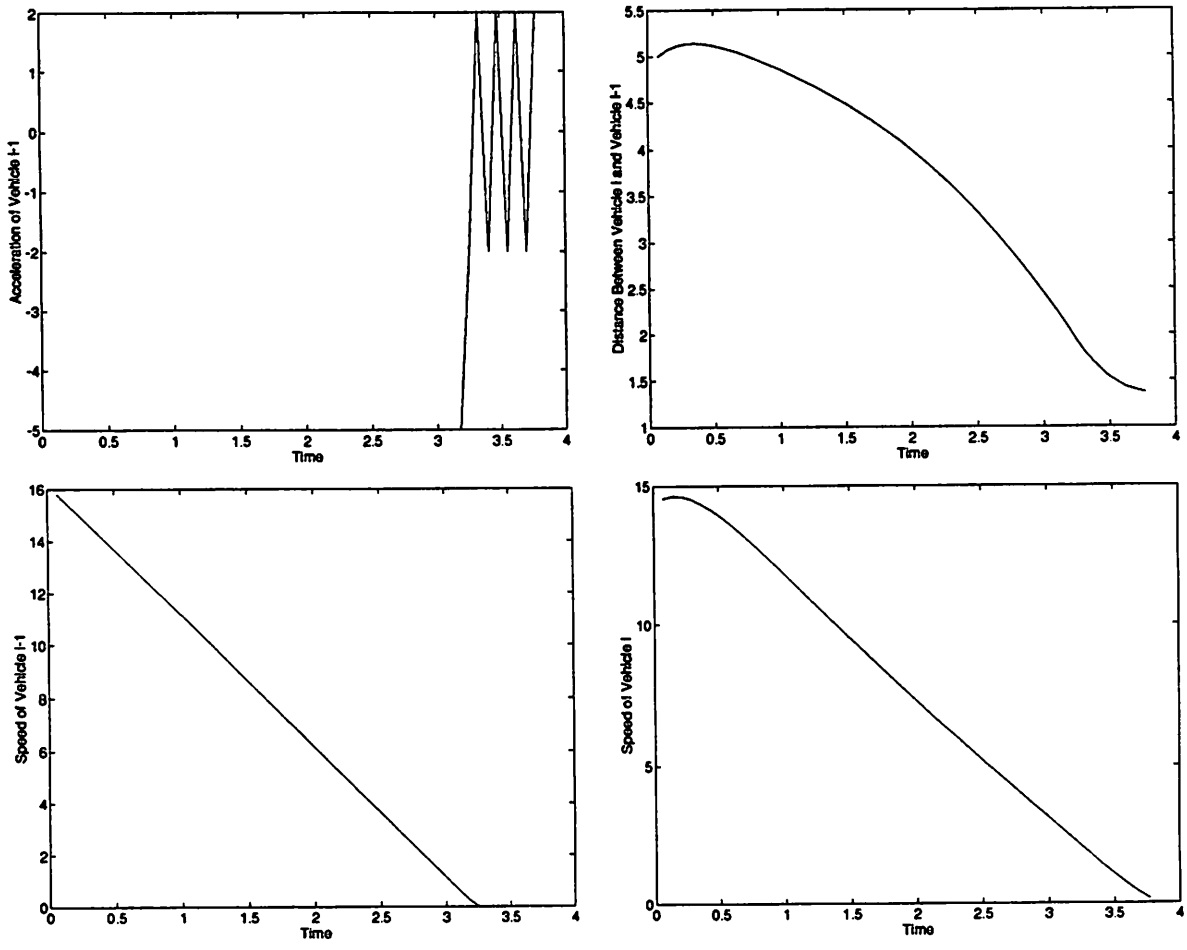

Figure 3.2: Solution of the Optimal Control Problem

problem:

$$
\begin{aligned}
\textit{Cost}: \quad & J = \min_t (d_{i-1} - d_i), \qquad (3.7)\\
\textit{Differential Eqn}: \quad & \dddot{d}_i = -3\ddot{d}_i - 3(\dot{d}_i - \dot{d}_{i-1}) + ((d_{i-1} - d_i) - (\dot{d}_i + 10)),\\
& \ddot{d}_{i-1} = u,\\
\textit{Initial Condition}: \quad & x(0) \in \mathcal{S}_L,\\
\textit{State Constraint}: \quad & \dot{d}_i \geq 0,\ \dot{d}_{i-1} \geq 0,\\
\textit{Control Constraint}: \quad & u \in [A_{Min}, A_{Max}]
\end{aligned}
$$

The optimal control problem is a free end time problem. The solution is obtained using mathematical programming techniques with the optimal cost $J = 1.4$, and

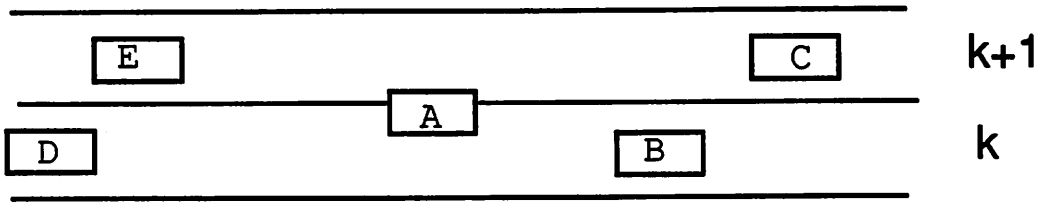

Figure 3.3: Change Lane Manuever

$x^*(0) = (5.0, 14.6, 15.8, 2.0)$.[2] Figure 3.2 shows the optimal control (acceleration of vehicle $i-1$), the distance between vehicle i and vehicle $i-1$, and the speed of vehicle $i-1$ and vehicle i respectively. The optimal control corresponds to vehicle $i-1$ applying full brakes for the first 3 seconds. Since the optimal cost $J = d_{i-1} - d_i$ is slightly above 1 meter, we conclude that starting from any initial condition in $\mathcal{S}_L$, despite any behaviour of vehicle $i-1$, the distance between the vehicles never falls below 1 metre. Therefore, vehicle i can safely switch to the control of equation 3.6 when $x(0) \in \mathcal{S}_L$.

Although in this example vehicle $i-1$ applies full brakes, in general, the trajectory which vehicle $i-1$ executes to minimize the separation between the two vehicles can be much more complicated. For example, vehicle $i-1$ can accelerate, causing vehicle i to accelerate, and then vehicle $i-1$ applies full brakes. The form of the optimal solution (the trajectory which vehicle $i-1$ executes to minimize the separation between the two vehicles) will depend on the control used by vehicle i.

To solve our problem we require a global optimum. This is in general difficult using mathematical programming techniques unless the problem is convex.

3.4 Changing Lane with Abstract Vehicles

In section 3.2, we described the design for a single lane. The basic maneuvers were the *merge* and the *split* maneuvers. In this section, we extend the design with the *change lane* maneuver that single vehicles execute to move from one lane to the next.

[2]The solution is due to Adam Schwartz.

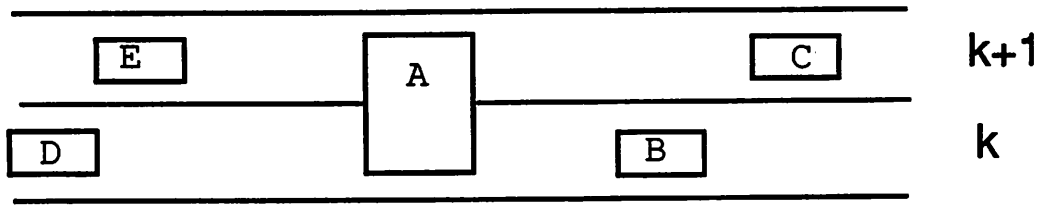

Figure 3.4: Changing Lane with Abstract Vehicles

The basic idea we use to show that the change lane maneuver is safe is the same as in section 3.3: a vehicle follows the vehicles in front at a safe distance and speed. Its control law prevents a high-speed collision with the vehicles in front, no matter what their behavior is. In the case of a single lane, the meaning of "front" is well-defined. In the case of a multilane highway, this is not so clear. Consider vehicle A changing from lane k to lane $k+1$ in figure 3.3. The process of changing lane takes a certain amount of time, and is a continuous phenomenon. It is not clear in figure 3.3, which vehicle is in front of vehicle A, and what should be the longitudinal control for vehicle A. Or when should the longitudinal control for D take B into account, rather than A.

Since keeping safely behind the vehicle in *front* was the key to proving safety in a single lane system, we extend this idea to multilane system by using the concept of an *abstract vehicle.* Abstract vehicles will be conceptual devices used to represent real vehicles. For example, a vehicle changing from lane k to lane $k+1$ will be represented by an abstract vehicle occupying both lane k and lane $k+1$ for some time. We will design the control system so that the abstract vehicles remain safe. Since a real vehicle is within the space occupied by the abstract vehicle, this will also guarantee the safety of real vehicles.

Consider figure 3.4. Vehicle A is changing from lane k to lane $k+1$, but it is represented by an abstract vehicle. The meaning of "front" is clear in this figure. Vehicles B and C both are in front of vehicle A, and vehicle A is in front of vehicles D and E. Vehicle A must remain a safe distance and at a safe speed behind vehicles B and C. Despite any erratic behavior on the part of vehicles B and C, the control law for vehicle A should prevent a collision with B or C. Of course if the abstract

vehicle is safe, then so is the real vehicle. The controls for vehicles D and E are identical to those in section 3.2, with vehicle A in front.

When a vehicle is ready to change lane, it turns on its change lane signal. At this time, it also becomes an abstract vehicle occupying two lanes. It switches to a longitudinal control which keeps the abstract vehicle safe from the vehicles in the front, and a lateral control which causes the vehicle to move from one lane to the next. The change lane signal also indicates to the other vehicles that this vehicle is an abstract vehicle occupying two lanes. The vehicles in the neighborhood take this into account when they figure out which vehicle is in front of them.

3.4.1 Longitudinal Control for an Abstract Vehicle

An abstract vehicle can have two vehicles in front of it—one in each lane— the objective of the longitudinal control should be to prevent a collision with either vehicle. Consider vehicles A,B and C in figure 3.4. The states of vehicles A,B and C are $x_A = (d_A, \dot{d}_A, \ddot{d}_A)$, $x_B = (d_B, \dot{d}_B, \ddot{d}_B)$ and $x_C = (d_C, \dot{d}_C, \ddot{d}_C)$ respectively, where d_A, d_B and d_C are the distances of the vehicles from the origin. The longitudinal control for vehicle A is

$$\dot{x}_A = C(x_A, x_B, x_C) \tag{3.8}$$

The unsafe set is $U_C = \{(x_A, x_B, x_C) | d_A = d_B \text{ or } d_A = d_C\}$ (i.e., A has a collision with B or C). An initial set S_C is specified such that U_C is unreachable when equation 3.8 starts from an initial state $(x_A, x_B, x_C) \in S_C$. To check that the control C satisfies this safety criterion, an optimal control problem can be solved as in section 3.3.

3.4.2 Change Lane Manuever

Suppose vehicle A wants to change from lane k to lane $k+1$. Let $[A]$ be the abstract vehicle which occupies both lane k and lane $k+1$. Since $[A]$ occupies two lanes, there may be a new vehicle in front of it in lane $k+1$. Furthermore, $[A]$ itself may be in front of another vehicle in lane $k+1$. Before A becomes an abstract vehicle, it must check that $[A]$ starts from a safe initial condition. And if $[A]$ is in

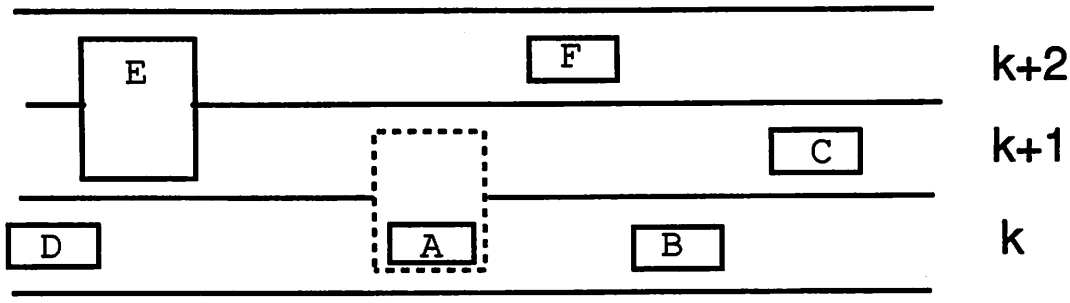

Figure 3.5: Change Lane Manuever

front of a new vehicle in lane $k+1$, then that vehicle must also start from a safe initial condition. Consider figure 3.5. Before A becomes an abstract vehicle, it must check that $(x_A, x_B, x_C) \in \mathcal{S}_C$. And since $[A]$ will be in front of E, it must also check that $(x_E, x_A, x_F) \in \mathcal{S}_C$. When these two conditions are true, A turns on its change lane signal, becomes an abstract vehicle, and begins to change lane. After A has finished changing lane, it turns off the change lane signal and resumes leader control.

Notice a subtle point in the design. Before A resumes leader control, it needs to check that the new initial condition is safe. Also, in figure 3.5, $[A]$ is safe from B, and D is safe from $[A]$. But when A finishes changing into lane $k+1$, D finds B in front of it, and may not be safe from it. We assume the controls satisfy the transitivity property (i.e., if A is safe from B, and D is safe from A, then D is safe from B) which rules out this possibility.

The possibility of two vehicles *simultaneously* turning on their change lane signals has to be avoided or resolved by using some coordination or contention resolution mechanism.

There is no high-speed collision involving a vehicle and the vehicle in front. This is also true for the abstract vehicles. Therefore, vehicles that are changing lane are also safe. Furthermore, a change in the highway configuration due to the beginning or ending of the change lane maneuver also keeps the system safe. Since *every* vehicle on the multilane AVHS is safe at all times, we conclude that the multilane AVHS is safe. It is interesting that the proof of safety in a multilane AVHS is independent of the lateral control law for the change lane maneuver.

3.5 Conclusion

We considered the problem of safety on an AVHS. We presented two main techniques: conservatively abstracting the dynamics of a vehicle by a simple differential inclusion; and representing a vehicle changing lane by an abstract vehicle occupying two lanes. Using these methods, it becomes possible to determine the safety of a vehicle by considering only its own controllers. When the controllers satisfy a set of constraints, the vehicle is safe. We showed that checking whether the controllers satisfy the set of constraints is equivalent to solving an optimal control problem. Since we prove that each vehicle is safe, we conclude the multilane AVHS operates safely.

Several problems need to be studied in more detail. Computing the initial set $\mathcal{S}_f$, and determining whether the unsafe set U_f is reachable from an initial condition in $\mathcal{S}_f$ is a key problem. Although the problem is equivalent to an optimal control problem, solving for the global optimal is difficult.

In the next chapter, we propose another solution to this problem. We provide computational methods to compute arbitrary close approximation of reach sets and invariant sets of differential equations and differential inclusions.

Chapter 4

ϵ-Approximation of Differential Inclusions

4.1 Introduction

A dynamical system $\dot{x} \in f(x)$ describes the flow of points in the space. Associated with a dynamical system are several interesting concepts: from an *invariant set*, points cannot escape; and a recurrent set is visited infinitely often. For the controlled system $\dot{x} = f(x, u)$, the question of whether there is a control $u \in U$ to steer the system from an initial state x_0 to a final state x_f is fundamental.

We approach the subject from the viewpoint of applications and an interest in computational methods. For the differential inclusion $\dot{x} \in f(x)$, we want to compute the invariant sets and the recurrent sets. For the controlled differential equation $\dot{x} = f(x, u)$, we want to determine the control $u \in U$ which steers the system from an initial state x_0 to a final state x_f. And we want to determine the reach set $Reach_f(X_0, [0, t])$ — the set of states that can be reached from the initial set of states X_0 within time t.

In this chapter, we propose a computational approach to solve some of these problems. For a Lipschitz differential inclusion $\dot{x} \in f(x)$ with initial set X_0, we propose a polyhedral method to obtain an arbitrary close approximation of $Reach_f(X_0, [0, t])$. For a differential inclusion $\dot{x} \in f(x)$, and any $\epsilon > 0$, we construct a *finite sample graph*

A^ϵ which has the property that every trajectory ϕ of $\dot{x} \in f(x)$ is also a "trajectory" in the graph A^ϵ. And every "trajectory" η of the finite graph A^ϵ has the property that $dist(\dot{\eta}(t), f(\eta(t))) \leq \epsilon$. Since A^ϵ is a finite graph, it can be analyzed using graph theoretic techniques. Using the finite graph A^ϵ, we can compute the *ϵ-invariant* sets of $\dot{x} \in f(x)$ — the sets which remain invariant under small perturbations in f.

In Section 4.2, we introduce our notation, and define the basic terms. In Section 4.3, we conservatively approximate the differential inclusion by a piecewise constant inclusion, and obtain an approximation of $Reach_f(X_0, [0,t])$. In Section 4.4, we obtain a finite graph A^ϵ from the differential inclusion $\dot{x} \in f(x)$, and use it to determine the properties of the differential inclusion. In Section 4.5, we discuss the application of techniques from Sections 4.3 and 4.4 to computing the ϵ-invariant sets and other properties of differential inclusions. In Section 4.6, we apply these methods to compute the invariant sets for two examples: a pendulum moving in the vertical plane, and the Lorenz equation. We also discuss procedures to improve the efficiency of our methods. In Section 4.7, we discuss possible directions for future work.

4.2 Preliminaries

Notation

$\mathcal{R}$ is the set of reals and $\mathcal{Z}$ is the set of integers. $B = \{x : |x| \leq 1\}$ is the unit ball. For sets $U, V \subset \mathcal{R}^n$, $U + V = \{u + v | u \in U \text{ and } v \in V\}$ and for $\alpha \in \mathcal{R}$, $\alpha U = \{\alpha u | u \in U\}$. For $\delta > 0$, $B_\delta(x)$ is the δ-ball centered at x, i.e., $B_\delta(x) = \{y : |y - x| \leq \delta\}$. For $X \subset \mathcal{R}^n$, $X_\epsilon = X + \epsilon B$.

For $x \in \mathcal{R}^n$, and $Y \subset \mathcal{R}^n$, the distance $dist(x, Y) = inf\{|x - y| : y \in Y\}$. For two sets $X, Y \subset \mathcal{R}^n$, the *Hausdorff distance* is $dist(X, Y) = inf\{r : X \subset Y + rB \text{ and } Y \subset X + rB\}$. Notice, that if $dist(X, Y) \leq \epsilon$, then for any $x \in X$, $dist(x, Y) \leq \epsilon$. For $X \subset \mathcal{R}^n$, $cl(X)$ is the closure of X, and $\overline{co}(X)$ is the smallest closed convex set containing X. For $X, Z \subset \mathcal{R}^n$, the restriction of X to Z is $X|_Z = X \bigcap Z$. For a set J, the complement of J is J^c. For sets X and Y, the difference $X \backslash Y = \{z | z \in X \text{ and } z \notin Y\}$.

A set-valued (multi-valued) function is $f : \mathcal{R}^n \longrightarrow \mathcal{R}^n$ where $f(x) \subset \mathcal{R}^n$. For a set-valued $f : \mathcal{R}^n \rightarrow \mathcal{R}^n$, the set-valued function $f_\epsilon : \mathcal{R}^n \rightarrow \mathcal{R}^n$ is given by $f_\epsilon(x) = f(x) + \epsilon B$. For $Z \subset \mathcal{R}^n$, $f(Z) = \bigcup_{x \in Z} f(x)$. We assume the infinity norm on $\mathcal{R}^n$ (i.e., $|x| = \max\{|x_1|, \ldots, |x_n|\}$).

Differential Inclusions

A differential inclusion is written as $\dot{x} \in f(x)$ where $f : \mathcal{R}^n \longrightarrow \mathcal{R}^n$ is a set-valued function. Differential inclusions can be used to model disturbances and uncertainties in the system. A differential equation $\dot{x} = f(x, u)$, where $u \in U$ is control or disturbance can be studied as the differential inclusion $\dot{x} \in g(x)$ where $g(x) = \{f(x, u) | u \in U\}$. The differential inclusion $\dot{x} \in g(x)$ captures every possible behaviour of f.

We say a differential inclusion $\dot{x} \in f(x)$ is Lipschitz with Lipschitz constant k provided $dist(f(x_1), f(x_2)) \leq k|x_2 - x_1|$. A trajectory $\phi : \mathcal{R} \longrightarrow \mathcal{R}^n$ is a solution of $\dot{x} \in f(x)$ provided $\dot{\phi}(t) \in f(\phi(t))$ *a.e.* We say f is convex-valued when $f(x)$ is convex for every x.

Definition 4.2.1 *For a differential inclusion $\dot{x} \in f(x)$ with initial set X_0, the set of states reached at time t is $Reach_f(X_0, t) = \{\phi(t) | \phi(0) \in X_0$ and ϕ is a solution of $\dot{x} \in f(x)\}$; the set of states reached upto time t is $Reach_f(X_0, [0, t]) = \bigcup_{\tau \in [0,t]} Reach_f(X, \tau)$, and the set of all states reached is $Reach_f(X, [0, \infty)) = \bigcup_t Reach_f(X, t)$.*

Example 4.2.1 *Consider the differential equation $\dot{x} = f(x) = -2x$, and $X_0 = [1, 2]$. Then $Reach_f(X_0, t) = [e^{-2t}, 2e^{-2t}]$, $Reach_f(X_0, [0, t]) = [e^{-2t}, 2]$ and $Reach_f(X_0, [0, \infty)] = (0, 2]$.*

There is a close relationship between the Lipschitz differential inclusion $\dot{x} \in f(x)$ and the convex-valued differential inclusion $\dot{x} \in \overline{co}(f(x))$. This is made by the following relaxation theorem [5, 42].

Theorem 4.2.1 *For a Lipschitz differential inclusion $\dot{x} \in f(x)$, $cl(Reach_f(X_0, t)) = Reach_{\overline{co}(f)}(X_0, t)$.*

Lemma 4.2.1 *If $\dot{x} \in f(x)$ is Lipschitz with constant k, then $\dot{x} \in f_\epsilon(x)$ is also Lipschitz with constant k.*

A solution η of the differential inclusion $\dot{x} \in f_\epsilon(x)$ has the property that $dist(\dot{\eta}(t), f(\eta(t))) \leq \epsilon$.

Example 4.2.2 *For the differential equation $\dot{x} = f(x) = -2x$, the differential inclusion $\dot{x} \in f_\epsilon(x) = [-2x - \epsilon, -2x + \epsilon]$. For $X_0 = [1, 2]$, the reach set $Reach_{f_\epsilon}(X_0, t) = [e^{-2t} + \epsilon(e^{-2t} - 1), 2e^{-2t} + \epsilon(1 - e^{-2t})]$ and for $\epsilon < 2$, $Reach_{f_\epsilon}(X_0, [0, \infty)] = (-\epsilon, 2]$.*

For further details on differential inclusions, see [6, 5, 42]. The following two results are obtained by using the Bellman-Gronwall inequality [18].

Lemma 4.2.2 *If $\dot{x} = f(x)$ is Lipschitz with constant k, and $y(t)$ and $z(t)$ are solutions of the differential equation, then*

$$|y(t) - z(t)| \leq |y(0) - z(0)|e^{kt}.$$

Lemma 4.2.3 *Let f, g be continuous functions such that $|f(x) - g(x)| < \epsilon$. If $\dot{x} = f(x)$ is Lipschitz with constant k, and $y(t)$ and $z(t)$ are solutions with $\dot{y}(t) = f(y(t))$ and $\dot{z}(t) = g(z(t))$, and $y(0) = z(0)$, then*

$$|y(t) - z(t)| \leq \frac{\epsilon}{k}(e^{kt} - 1).$$

Grids and Graphs

For $\beta > 0$, define the β-grid in $\mathcal{R}^n$ to be the set $G = ((\mathcal{Z} + \frac{1}{2})\beta)^n$ where $((\mathcal{Z} + \frac{1}{2})\beta) = \{\cdots, -\frac{3}{2}\beta, -\frac{1}{2}\beta, \frac{1}{2}\beta, \frac{3}{2}\beta, \cdots\}$. For $x \in \mathcal{R}^n$, define the quantization of x as $[x] = g$ where g is the nearest grid point (i.e., $g \in G$ and $|x - g| \leq \frac{\beta}{2}$). For $g \in G$, define $\langle g \rangle = \{x : [x] = g\}$. For the grid in figure 4.1, $(\frac{1}{2}\beta, \frac{1}{2}\beta) \in G$, and $\langle(\frac{1}{2}\beta, \frac{1}{2}\beta)\rangle = [0, \beta] \times [0, \beta]$. Notice, some points maybe equi-distant from two grid points. For example, $[(0, \frac{1}{2}\beta)] = (\frac{1}{2}\beta, \frac{1}{2}\beta)$ and $[(0, \frac{1}{2}\beta)] = (-\frac{1}{2}\beta, \frac{1}{2}\beta)$.

Definition 4.2.2 *Given a β-grid, a differential inclusion $\dot{x} \in f(x)$, a sampling time Δ, and an initial set $X_0 \subset \mathcal{R}^n$, define the sampled trajectories $Traj_f(X_0) = \{([\phi(m\Delta)])_{m \in \mathcal{Z}^+} : \phi(0) \in X_0$, and $\dot{\phi}(t) \in f(\phi(t))\ \}$.*

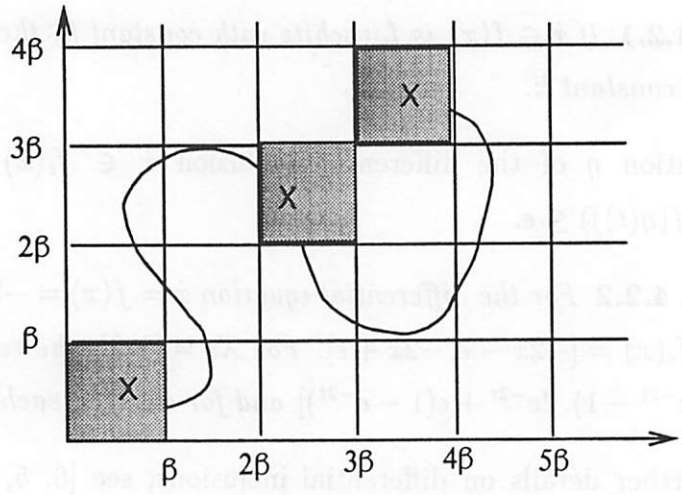

Figure 4.1: A β-grid and a sample trajectory

The trajectories of $\dot{x} \in f(x)$ are sampled every Δ time units and then quantized to obtain the sampled trajectories $Traj_f(X_0)$.

Example 4.2.3 *For the differential equation* $\dot{x} = -2x$ *and* $X_0 = [1,2]$, $Traj_f(X_0) = \{([x(0)e^{-2m\Delta}])_{m\in\mathcal{Z}^+} : x(0) \in [1,2]\}$.

Figure 4.1 shows a β-grid and a trajectory. The "crosses" on the trajectory mark the points that are sampled every Δ time units. The sequence of grids in which the "crosses" appear is recorded, and forms the sampled trajectory. Notice, it is not the sampled value that is recorded, but the grid in which the sample point appears that is recorded.

Definition 4.2.3 *A directed graph is* $A = (V, E)$ *where* V *is the set of vertices and* $E \subset V \times V$ *is the set of edges. A path* $\pi = v_0 v_1 \ldots v_n$ *where* $(v_i, v_{i+1}) \in E$. *For a set* $W \subset V$, $Reach_A(W) = \{v_n | \pi = v_0 \ldots v_n$ *is a path and* $v_0 \in W\}$.

The set $Reach(W)$ is the set of vertices of the graph that can be reached from the vertices in W. Figure 4.2 is a directed graph with vertices $V = \{A, B, C, D, E, F\}$. The set $Reach(\{C\}) = \{A, B, C, D\}$. A set $S \subset V$ is a *strongly connected set* provided for any vertices $v, w \in S$, there is a path in S from v to w. In the graph of Figure 4.2, $\{B, C, D\}$ is a strongly connected set. For a graph $A = (V, E)$ and $M \subset V$, the

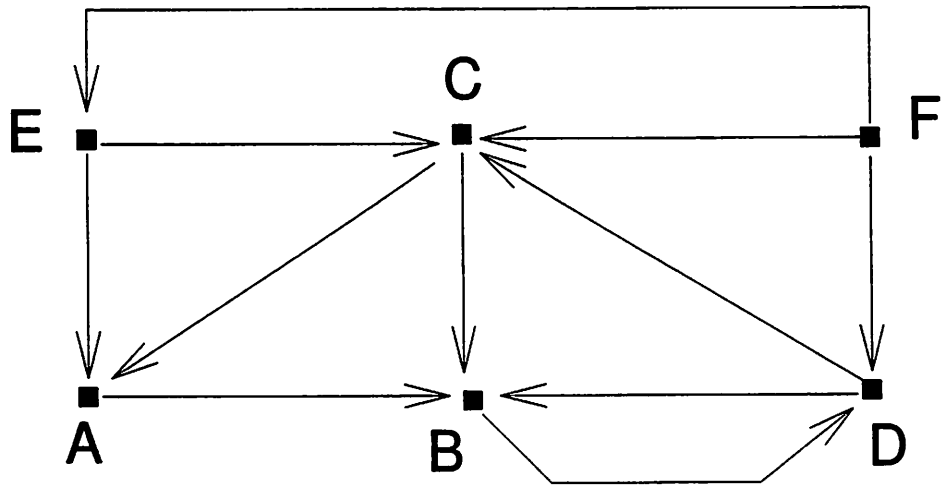

Figure 4.2: A Directed Graph

subgraph $(A)_M$ is obtained by deleting all vertices not in M and all edges whose endpoints are not in M. For a graph $A = (V, E)$, $Reach(W)$ and the strongly connected sets of the graph can be computed using graph search algorithms such as depth-first search or breadth-first search.

Definition 4.2.4 *For a graph* $A = (V, E)$ *and a set* $X_0 \subset V$, *define* $Traj_A(X_0) = \{(q_i)_{i \in Z^+} : q_0 \in X_0$ *and* $(q_i, q_{i+1}) \in E\}$.

A possible trajectory in the graph of figure 4.2 starting from vertex A is $(ABDC)^\omega$. The notation $\sigma^\omega = \sigma\sigma\ldots$ means the string σ is repeated forever.

4.3 Computing the Reach Set of Differential Inclusions

In this section, we give a method to compute the reach set of a convex-valued Lipschitz differential inclusion $\dot{x} \in f(x)$. We conservatively abstract the differential inclusion by a piecewise constant inclusion, and then compute the reach set of the piecewise constant inclusion. As a result, we overestimate the reach set of the original differential inclusion. We show the error in the estimate can be made arbitrarily small by abstracting the differential inclusion arbitrarily closely.

Approximating by Piecewise Constant Differential Inclusions

A convex polyhedron is a bounded set defined by a set of linear inequalities. Define $\mathcal{P}$ to be the set of all convex polyhedra. We will approximate a convex valued differential inclusion $\dot{x} \in f(x)$ on a region R where $R \in \mathcal{P}$. Let $C = \{c_1, \ldots, c_k\}$ be a cover of R where each $c_j \in \mathcal{P}$. We say C is a δ-cover provided for every $x \in R$, $B_\delta(x)|_R \subset c_i$ for some i. The δ-cover property states that the δ-ball around each point x, when restricted to R, is completely contained within some c_i.

Let C be a δ-cover of R. Associate with C a collection $D = \{d_1, \ldots, d_k\}$ such that for each $c_i \in C$, $d_i \in \mathcal{P}$, $f(c_i) \subset d_i$ and $dist(f(x), d_i) \leq \epsilon$ for all $x \in c_i$. We say $\langle C, D \rangle$ is an ϵ-approximation of the differential inclusion $\dot{x} \in f(x)$.

Construction to form the cover:
To obtain an ϵ-approximation of the convex-valued Lipschitz differential inclusion $\dot{x} \in f(x)$ with Lipschitz constant k, define a $\frac{\epsilon}{6k}$-grid G. The cover is $C = \{c_g | g \in G$ where $\{c_g : |x - g| \leq \frac{\epsilon}{4k}\}\}$. First, notice the cover satisfies the δ-cover property for $\delta = \frac{\epsilon}{6k}$ (since for any x, $B_\delta(x) \subset c_g$ where $g = [x]$). With c_g, we associate the constant inclusion $f(g) + \frac{\epsilon}{4}B$. First note that for $x \in c_g$, $f(x) \subset f(g) + k|x - g|B$. Thus $f(x) \subset f(g) + \frac{\epsilon}{4}B$. Similarly $f(g) \subset f(x) + \frac{\epsilon}{4}B$. Therefore $dist(f(x), f(g) + \frac{\epsilon}{4}B) \leq \frac{\epsilon}{2}$. Since $f(g) + \frac{\epsilon}{4}B$ is convex, it can be approximated by a polyhedron d_g such that $f(x) \subset d_g$ and $dist(f(x), d_g) \leq \epsilon$ for all $x \in c_g$. We get a finite cover for R by restricting C to R. Using the above construction, we can approximate any convex valued lipschitz differential inclusion $\dot{x} \in f(x)$ by an ϵ-approximation $\langle C, D \rangle$.

Figure 4.3 shows a $\frac{\epsilon}{6k}$ grid. One of the grid point is indicated by the letter "g". The cover c_g is indicated by the bold square. Note, for $x \in \langle g \rangle$, $\mathcal{B}_\delta(x) \subset c_g$ for $\delta \leq \frac{\epsilon}{6k}$.

Example 4.3.1 *Consider the following differential inclusion:*

$$\dot{x}_1 = 2x_1 - x_2$$

$$\dot{x}_2 \in x_1 x_2 + [-1, +1].$$

We are interested in the region $R = [-2, +2] \times [-2, +2]$. We first determine the Lipschitz constant for the differential inclusion. For $x, y \in R$, suppose $\dot{y} \in f(y)$.

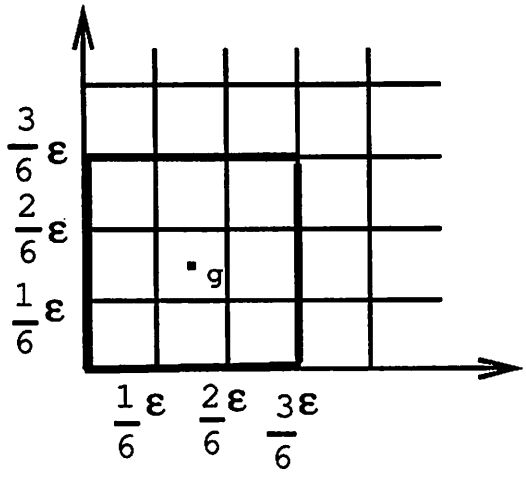

Figure 4.3: A $\frac{\epsilon}{6k}$-grid

Then for some $\dot{x} \in f(x)$,

$$|\dot{y}_1 - \dot{x}_1| \leq |2(y_1 - x_1) - (y_2 - x_2)| \leq 3|y - x|$$

and

$$|\dot{y}_2 - \dot{x}_2| \leq |y_1 y_2 - x_1 x_2| \leq |x_2(y_1 - x_1) + y_1(y_2 - x_2)| \leq 4|y - x|.$$

Thus

$$|\dot{y} - \dot{x}| \leq \max\{|\dot{y}_1 - \dot{x}_1|, |\dot{y}_2 - \dot{x}_2|\} \leq 4|y - x|.$$

Therefore the Lipschitz constant $k = 4$ *and*

$$f(y) \subset f(x) + 4B|y - x|.$$

To obtain an ϵ*-approximation, we define a grid* G *of size* $\frac{\epsilon}{6(4)}$, *and for* $g \in G$, *we define* $c_g = \{x : |x - g| \leq \frac{\epsilon}{4(4)}\}$. *The constant inclusion for* c_g *is*

$$f(g) + \frac{\epsilon}{4}B = \begin{pmatrix} 2g_1 - g_2 + [-\frac{\epsilon}{4}, +\frac{\epsilon}{4}] \\ g_1 g_2 + [-1 - \frac{\epsilon}{4}, +1 + \frac{\epsilon}{4}] \end{pmatrix}.$$

Since $f(g) + \frac{\epsilon}{4}B$ *is a polyhedron, we define* $d_g = f(g) + \frac{\epsilon}{4}B$.

Reach Set Computation

Suppose C is a δ-cover of R and $\langle C, D\rangle$ is an ϵ-approximation of the differential inclusion $\dot{x} \in f(x)$. In this section we obtain an approximation of $Reach_f(X_0, t)$ using the ϵ-approximation $\langle C, D\rangle$.

Definition 4.3.1 *For $W \subset R$, define $Next(W) = \bigcup_k \{y \in c_k | y = x + t\alpha$ for $x \in c_k \bigcap W, \alpha \in d_k$, and $t \geq 0$ }.*

For a set of states W, $Next(W)$ is the set of states that is reached after some time. For each $c_k \in C$, $Next(W)$ comprises states that are reached from $c_k \bigcap W$ by following a direction in d_k and remaining inside c_k.

Construction for Next(W):
For a polyhedron w, and a direction polyhedron d, define $ext(w, d) = \{w + t\alpha : \alpha \in d$ and $t \geq 0$ } (i.e., all the states reached from w by moving in a direction in d). Notice that $ext(w, d)$ is a polytope (i.e, defined by a set of linear inequalities). Next suppose $W = \bigcup_i w_i$ where each w_i is a polyhedron. Then $W = \bigcup_i \bigcup_j (w_i \bigcap c_j)$ since C is a cover of P. We get $Next(W) = \bigcup_i \bigcup_j (ext(w_i \bigcap c_j, d_j) \bigcap c_j)$. The construction implies that when W is a union of polyhedra, $Next(W)$ is also a union of polyhedra.

Lemma 4.3.1 *If V is a closed convex set and $f : [0, T] \longrightarrow V$, then $\frac{1}{T} \int_0^T f(t)dt \in V$.*

Let M be the maximum value of $|f|$ on R. Using Lemma 4.3.1 and the δ-cover property, we show that $Reach_f(W, [0, \frac{\delta}{M}]) \subset Next(W)$.

Theorem 4.3.1 *If $\dot{x} \in f(x)$ is a differential inclusion, M is the maximum value of $|f|$ on R, and C is a δ-cover of R then $Reach_f(W, [0, \frac{\delta}{M}]) \subset Next(W)$.*

Proof: Suppose $y \in Reach_f(W, [0, \frac{\delta}{M}])$. Then $y = \phi(\lambda)$, where ϕ is a trajectory of the differential inclusion $\dot{x} \in f(x)$ over interval $[0, \lambda]$, $\lambda \leq \frac{\delta}{M}$, starting at $\phi(0) \in W$. For $t \in [0, \lambda]$, $|\phi(t) - \phi(0)| \leq tM \leq \lambda M \leq \frac{\delta}{M} M = \delta$. From the δ-cover property, there is a j such that $\phi(t) \in c_j$ and $\dot{\phi}(t) \in d_j$ *a.e* for $t \in [0, \lambda]$. Thus

$\phi(\lambda) - \phi(0) = \int_0^\lambda \dot{\phi}(t)dt = \lambda\alpha_j$ where $\alpha_j \in d_j$ (from Lemma 4.3.1). Therefore $y = \phi(\lambda) \in Next(W)$. ∎

Notice, we require C to be a δ-cover to obtain the "time advancing" property in Theorem 4.3.1, i.e., the trajectory stays in c_j for time at least $\frac{\delta}{M}$. The reach set can be computed using the following iteration.

Reach Set Computation:

$$R_0 = X_0$$
$$R_{i+1} = Next(R_i) \bigcup R_i$$

Assume X_0 is a union of polyhedra. From the previous discussion each R_i is a union of polyhedra. When R_i is a union of polyhedra, we can compute $Next(R_i)$, and hence R_{i+1}. From Theorem 4.3.1, each iteration of the algorithm advances the time by at least $\frac{\delta}{M}$ units. To compute $Reach_f(W, [0,t])$ requires at most $l = \lceil \frac{Mt}{\delta} \rceil$ iterations (i.e., $Reach_f(W, [0,t]) \subset R_l$).

In general, R_i will be a proper subset of R_{i+1}. But if $R_i = R_{i+1}$, then the *Reach Set Computation* terminates. In this case, as the following theorem shows, we get an approximation of the infinite time reachable set $Reach_f(X_0, [0,\infty))$ in a finite number of steps.

Theorem 4.3.2 *If $\langle C, D\rangle$ is an ϵ-approximation of the differential inclusion $\dot{x} \in f(x)$, and $R_i = R_{i+1}$ in the* Reach Set Computation, *then $Reach_f(X_0, [0,\infty)) \subset R_i \subset Reach_{f_\epsilon}(X_0, [0,\infty))$. Furthermore R_i is invariant under $\dot{x} \in f(x)$.*

The *Reach Set Computation* procedure can also be used to compute the reach set at a specific time t. This is done by augmenting the state space to $y = (x, \tau)$ where $\dot{y} \in h(y) = (f(x), \{1\})$ (i.e., $\dot{x} \in f(x)$ and $\dot{\tau} = 1$). The variable τ keeps track of the time. The ϵ-approximation $\langle C^h, D^h\rangle$ of $\dot{y} \in h(y)$ is obtained from the ϵ-approximation $\langle C, D\rangle$ of $\dot{x} \in f(x)$ by defining $C^h = \{c_g \times [0, Max] : c_g \in C\}$ where Max is the largest time for which we want to compute the reach set, and $D^h = \{d_g \times \{1\} | d_g \in D\}$. Using the *Reach Set Computation* on the augmented system, we can compute R_l where $l = \lceil \frac{Mt}{\delta} \rceil$, and $Reach_h(W \times \{0\}, [0,t]) \subset R_l$. Define

$R_l(\tau = t)$ to be the intersection of R_l with the hyperplane $\tau = t$, and $R_l(\tau \leq t)$ to be the intersection of R_l with the half-space $\tau \leq t$. Clearly $Reach_f(W,t) \subset R_l(\tau = t)$, and $Reach_f(W,[0,t]) \subset R_l(\tau \leq t)$.

Example 4.3.2 *Consider the differential equation $\dot{x} = f(x) = -2x$ with initial condition $x(0) = x_0$. The augmented system is $y = (x,\tau)$ with $\dot{y} = h(y) = (-2x, 1)$ with initial condition $y(0) = (x_0, 0)$. The reach set of inclusion $\dot{y} \in h(y)$ is $Reach_h(\{(x_0,t)\},[0,\infty)) = \{(x_0e^{-2t},t)|t \geq 0\}$. Intersecting with the hyperplane $(\tau = t)$, we get (x_0e^{-2t},t) — the reach set of differential equation f at time t.*

Lemma 4.3.2 *If $\langle C, D\rangle$ is an ϵ-approximation of the differential inclusion $\dot{x} \in f(x)$, then for $l = \lceil \frac{Mt}{\delta} \rceil$, $Reach_f(X_0,t) \subset R_l(\tau = t) \subset Reach_{f_\epsilon}(X_0,t)$.*

We also want to get a bound on the error in the approximation $dist(Reach_f(X_0,t), R_l(\tau = t))$. The following theorem shows that the error can be made arbitrarily small.

Theorem 4.3.3 *Suppose $\dot{x} \in f(x)$ is a Lipschitz differential equation with Lipschitz constant k. Then for any $\gamma > 0$, and any $t \geq 0$, using the Reach Set Computation procedure, we can compute R_l as a union of polyhedra such that $Reach_f(X_0,t) \subset R_l(\tau = t)$ and $dist(Reach_f(X_0,t), R_l(\tau = t)) < \gamma$.*

Proof: Given $t > 0$ and $\gamma > 0$, choose $\epsilon = \frac{\gamma k}{(e^{kt}-1)}$ where k is the lipschitz constant. In the augmented system, for $l \geq \lceil \frac{4Mtk}{\epsilon} \rceil$, $Reach_f(X_0,t) \subset R_l(\tau = t)$. For $y \in R_l(\tau = t)$, there is a trajectory ϕ with $\phi(0) \in X_0$ and $\phi(t) = y$ such that $\dot{\phi}(\alpha) \in f_\epsilon(\phi(\alpha))$ *a.e* (from construction of $R_l(\tau = t)$). Using Lemma 4.2.3, we get $dist(Reach_f(X_0,t), R_l(\tau = t)) \leq \frac{\epsilon(e^{kt}-1)}{k} = \gamma$. ∎

We can compute $Reach_f(X_0,[0,t])$ with the same error using $R_l(\tau \leq t)$.

In this section we discussed only convex-valued Lipschitz differential inclusions. But from the relaxation theorem (theorem 4.2.1), for a Lipschitz differential inclusion $\dot{x} \in f(x)$, it suffices to study the convex-valued differential inclusion $\dot{x} \in \overline{co}(f(x))$.

The *Reach Set Computation* procedure we described can be automated using computer tools available for analysis of hybrid systems [2, 3, 17, 31, 29]. An equivalent hybrid automaton can be constructed by associating location l_g with c_g, differential inclusion d_g with location l_g, and guard $c_g \bigcap c_h$ with the edge from location l_g to l_h. See [2, 3, 17, 31, 30, 29] for more details on hybrid systems and their analysis.

We used polyhedral inclusions to approximate the differential inclusion $\dot{x} \in f(x)$. Instead, a decidable class of hybrid systems such as [30, 17], can be used to approximate a differential inclusion and prove the same result as Theorem 4.3.3 [7]. The decidable hybrid systems have the intersting property that the infinite time reachable set for them can be computed in a finite number of steps. In the next section, we provide another method which can be used to compute the infinite time reachable set in a finite number of steps.

4.4 Sample Graph Approximation

In this section, we prove the following result: given a Lipschitz differential inclusion $\dot{x} \in f(x)$, for any ϵ we can find a finite graph A^ϵ such that $Traj_f(X_0) \subset Traj_{A^\epsilon}([X_0]) \subset Traj_{f_\epsilon}(X_0)$. That is, A^ϵ contains the trajectories of $\dot{x} \in f(x)$, and the trajectories of A^ϵ are contained within the trajectories of $\dot{x} \in f_\epsilon(x)$. As a consequence, we get that for every $\epsilon > 0$, there is a finite graph A^ϵ such that for every "trajectory" η of A^ϵ, $dist(\dot{\eta}(t), f(\eta(t))) \leq \epsilon$. Using this, we can compute the ϵ-invariant sets of the inclusion $\dot{x} \in f(x)$. These are sets which remain invariant under ϵ-perturbations of f. We also use the sample graph A^ϵ to find other properties of the differential inclusion $\dot{x} \in f(x)$.

Definition 4.4.1 *Given a differential inclusion $\dot{x} \in f(x)$, and a sampling time Δ, define the map $S_f : \mathcal{R}^n \to \mathcal{R}^n$ with $S_f(y) = Reach_f(\{y\}, \Delta)$.*

Note that when f is a differential inclusion, S_f is a set-valued map. The map S_f samples the trajectory every Δ time units. Instead of working with the differential inclusion $\dot{x} \in f(x)$, we will work with the discrete dynamical system $x_{k+1} \in S_f(x_k)$.

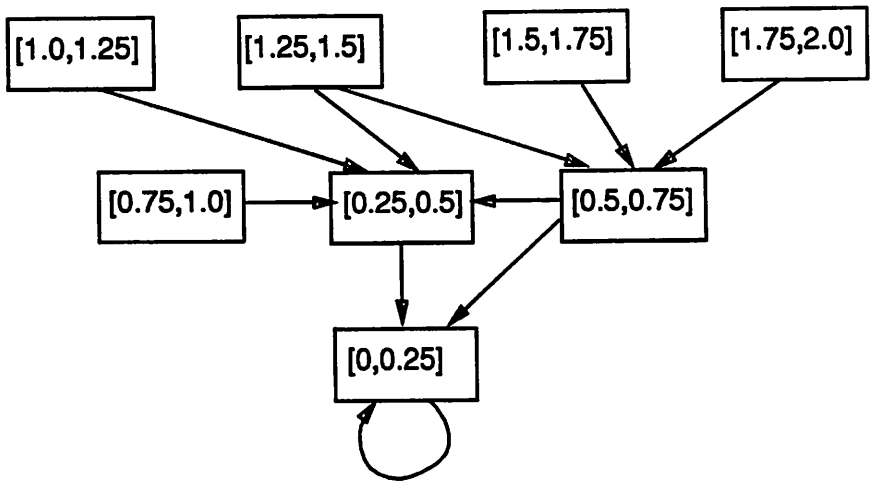

Figure 4.4: The Sample Graph of $\dot{x} = -2x$

Example 4.4.1 *For the differential equation* $\dot{x} = f(x) = -2x$, $S_f(y) = ye^{-2\Delta}$.

Sample Graph Construction:
For a differential inclusion $\dot{x} \in f(x)$ *and the* β*-grid, we construct the* sample graph. *The vertices of the sample graph are the grid points* $((\mathcal{Z} + \frac{1}{2})\beta)^n$, *and there is an edge from* $[x]$ *to* $[y]$ *provided* $y \in S_f(x)$. *That is, there is an edge from vertex* g *to vertex* h *provided there is a trajectory which takes some* $x \in \langle g \rangle$ *to some* $y \in \langle h \rangle$. *More formally, the sample graph is* $A = (V, E)$ *where* $V = ((\mathcal{Z} + \frac{1}{2})\beta)^n$ *are the vertices, and* $E \subset V \times V$ *are the edges with* $(g, h) \in E$ *for* $g, h \in V$ *provided* $S_f(\langle g \rangle) \bigcap \langle h \rangle \neq \emptyset$ *(i.e,* $y \in Reach_f(\langle g \rangle, \Delta)$ *for some* $y \in \langle h \rangle$*). When we are interested in studying the differential inclusion on a bounded region, we restrict the graph* A *to the bounded region and get a finite graph.*

Example 4.4.2 *For the differential equation* $\dot{x} = -2x$, *figure 4.4 shows the sample graph for interval* $[0, 2]$ *where the sample time* $\Delta = 0.5$ *and grid separation is* $\beta = 0.25$.

To get an ϵ-approximation of the differential equation $\dot{x} \in f(x)$, we need to construct the sample graph from a sufficiently small grid. The following theorem will enable us to choose an appropriate grid for a given ϵ.

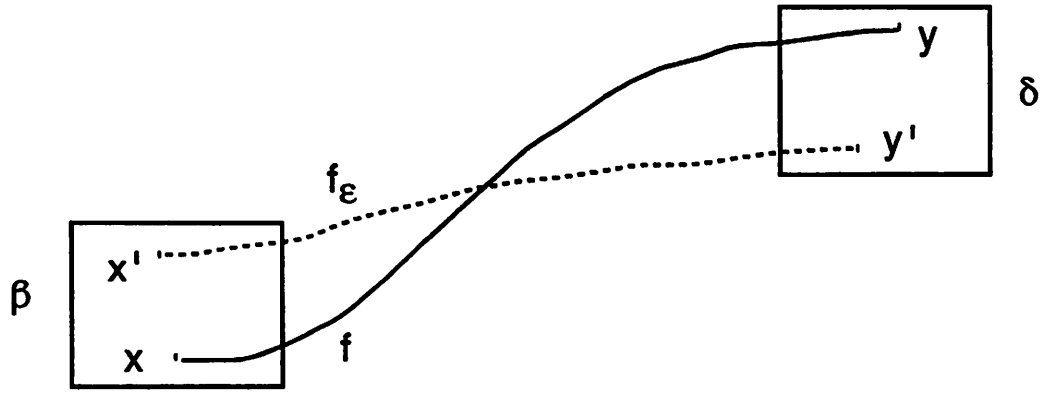

Figure 4.5: Trajectories of $\dot{x} \in f(x)$ and $\dot{x} \in f_\epsilon(x)$

Theorem 4.4.1 *If f is a Lipschitz differential inclusion with Lipschitz constant k, $|x-x'| \leq \beta$, $|y-y'| \leq \delta$, and $y \in S_f(x)$ with sample time Δ, then for $\epsilon \geq (k+\frac{1}{\Delta})(\delta+\beta)$, $y' \in S_{f_\epsilon}(x')$ (figure 4.5).*

Proof: Suppose $\alpha : [0, \Delta] \longrightarrow \mathcal{R}^n$ is a trajectory for which $\alpha(0) = x$, $\alpha(\Delta) = y$, and $\dot{\alpha}(t) \in f(\alpha(t))$. We define the trajectory $\eta : [0, \Delta] \longrightarrow \mathcal{R}^n$ by

$$\eta(t) = \alpha(t) + \frac{t}{\Delta}(y' - y) + \frac{(\Delta - t)}{\Delta}(x' - x).$$

Note that $\eta(0) = x'$, $\eta(\Delta) = y'$, and

$$|\dot{\eta}(t) - \dot{\alpha}(t)| \leq \frac{(\delta + \beta)}{\Delta} \tag{4.1}$$

$$|\eta(t) - \alpha(t)| \leq (\delta + \beta) \tag{4.2}$$

We will show $\dot{\eta}(t) \in f_\epsilon(\eta(t))$. From equation 4.1, it follows that

$$\dot{\eta}(t) \in f(\alpha(t)) + \frac{(\delta + \beta)}{\Delta} B$$

Since f is Lipschitz, from equation 4.2 we get

$$f(\alpha(t)) \subset f(\eta(t)) + k(\delta + \beta)B$$

Therefore

$$\dot{\eta}(t) \in f(\eta(t)) + (k + \frac{1}{\Delta})(\delta + \beta)$$

Thus $\dot{\eta}(t) \in f_\epsilon(\eta(t))$ when $\epsilon \geq (k + \frac{1}{\Delta})(\delta + \beta)$. ∎

We can show, using the above theorem, that for any $\epsilon > 0$, we can find a β-grid such that $Traj_f(X_0) \subset Traj_{A^\epsilon}([X_0]) \subset Traj_{f_\epsilon}(X_0)$ where A^ϵ is the finite graph obtained from the *sample graph construction.*

Theorem 4.4.2 *If $\dot{x} \in f(x)$ is a Lipschitz differential inclusion with Lipschitz constant k, and $\epsilon > 0$, then for $\beta \leq \frac{\epsilon}{(k+\frac{1}{\Delta})}$, $Traj_f(X_0) \subset Traj_{A^\epsilon}([X_0]) \subset Traj_{f_\epsilon}(X_0)$ where A^ϵ is the sample graph on the β-grid.*

Proof: By construction of graph A^ϵ on the β-grid, $Traj_f(X_0) \subset Traj_{A^\epsilon}([X_0])$. Next suppose $q = (q_0, q_1, \ldots) \in Traj_{A^\epsilon}([X_0])$. Then for each i, there are $x_i \in \langle q_i \rangle$ and $y_i \in \langle q_i \rangle$ such that for $i \geq 1$, $y_i \in S_f(x_{i-1})$ (define $y_0 = x_0$). From theorem 4.4.1 $y_i \in S_{f_\epsilon}(y_{i-1})$ (by choosing $\delta = 0$) for a grid with $\beta \leq \frac{\epsilon}{k+\frac{1}{\Delta}}$. We can construct a trajectory y with $\dot{y}(t) \in f_\epsilon(y(t))$ and $y(j\Delta) = y_j \in \langle q_j \rangle$. Therefore $Traj_A([X_0]) \subset Traj_{f_\epsilon}(X_0)$. ∎

Modification of Sample Graph Construction: The construction of the graph A^ϵ required us to compute $S_f(\langle g \rangle)$. This is not necessary to prove the approximation result of Theorem 4.4.2. Instead, a conservative approximation of $S_f(\langle g \rangle)$ can be made. We do this either with the methods of Section 4.3 using Theorem 4.3.3 or by using Lemma 4.2.2. A conservative approximation of $S_f(\langle g \rangle)$ is made by computing $W(\langle g \rangle)$ where $S_f(\langle g \rangle) \subset W(\langle g \rangle)$, and the error $dist(S_f(\langle g \rangle), W(\langle g \rangle)) < \zeta$. We use $W(\langle g \rangle)$ instead of $S_f(\langle g \rangle)$ to form the graph A^ϵ. That is, the sample graph is $A^\epsilon = (V, E)$ where $V = ((\mathcal{Z} + \frac{1}{2})\beta)^n)$ are the vertices, and $(g, h) \in E$ for $g, h \in V$ provided $W_f(\langle g \rangle) \bigcap \langle h \rangle \neq \emptyset$. Using Theorem 4.4.1, we obtain the same result as Theorem 4.4.2 when the β-grid is chosen so that $\epsilon \geq (k + \frac{1}{\Delta})(\beta + \zeta)$.

Example 4.4.3 *For a differential equation $\dot{x} = f(x)$ with Lipschitz constant k, we construct a sample graph on a β-grid using the modified Sample Construction method. We note that $S_f(\langle g \rangle) \subset S_f(\{g\}) + \frac{\beta}{2} e^{k\Delta} B$ from Lemma 4.2.2. For $\Delta = \frac{\ln 2}{k}$, $S_f(\langle g \rangle) \subset S_f(\{g\}) + \beta B$. Hence, for $\epsilon \geq 2(k + \frac{1}{\Delta})\beta$, $Traj_f(X_0) \subset Traj_{A^\epsilon}([X_0]) \subset Traj_{f_\epsilon}(X_0)$.*

The trajectories of finite graph A^ϵ contain information about the trajectories of the differential inclusion $\dot{x} \in f(x)$. From any trajectory (q_k) of A^ϵ, we can obtain a continuous trajectory ϕ_q such that $[\phi_q(k\Delta)] = q_k$ and $dist(\dot{\phi}_q(t), f(\phi_q(t))) \leq \epsilon$. Because we can "sandwich" the trajectories of A^ϵ between the trajectories of the differential inclusion $\dot{x} \in f(x)$ and $\dot{x} \in f_\epsilon(x)$, we can obtain useful results about the invariant sets and the recurrent sets of $\dot{x} \in f(x)$ from the graph A^ϵ.

Using the finite sample graph A^ϵ, we can get an approximation of the infinite time reachable set $Reach_f(X_0, [0, \infty))$.

Theorem 4.4.3 *For a Lipschitz differential equation, and any* $\epsilon > 0$, $Reach_f(X_0, [0, \infty)) \subset W \subset Reach_{f_\epsilon}(X_0, [0, \infty))$ *where* $W = Reach_f(\langle Reach_{A^\epsilon}([X_0])\rangle, [0, \Delta])$ *is an invariant set of* $\dot{x} \in f(x)$.

Notice the similarity between Theorem 4.3.2 and Theorem 4.4.3. The difference is that $Reach_{A^\epsilon}([X_0])$ in Theorem 4.4.3 is computed in a finite number of steps. We will discuss other advantages and disadvantages of the polyhedral approach vs. the graph approach in Section 4.6. We discuss a small technicality: the set W satisfying $Reach_f(X_0, [0, \infty)) \subset W \subset Reach_{f_\epsilon}(X_0, [0, \infty))$ in Theorem 4.4.3 can be computed in a finite number of steps by using the sample graph $A^{\frac{\epsilon}{2}}$ for inclusion $\dot{x} \in f_{\frac{\epsilon}{2}}(x)$, and then using an $\frac{\epsilon}{2}$-approximation of $\dot{x} \in f(x)$ to compute W using the method of Section 4.3.

The sample graph for the continuous system was created by looking at the discrete system $x_{m+1} \in S_f(x_m)$. We next state our theorem directly for discrete systems. For the discrete system $x_{m+1} \in g(x_m)$ where g is lipschitz with constant k, define $Traj_g(X_0) = \{([\alpha(m)])_{m \in Z^+} | \alpha(m) \in g(\alpha(m-1))$ and $\alpha(0) \in X_0\}$. The sample graph A^ϵ is created on the β-grid by putting an edge from vertex j to vertex h provided $g(\langle j \rangle) \cap \langle h \rangle \neq \emptyset$.

Theorem 4.4.4 *If* $x_{m+1} \in g(x_m)$ *is a discrete system where* g *is lipschitz with constant* k, *then for* $\beta \leq \frac{\epsilon}{k}$, $Traj_g(X_0) \subset Traj_{A^\epsilon}([X_0]) \subset Traj_{g_\epsilon}(X_0)$.

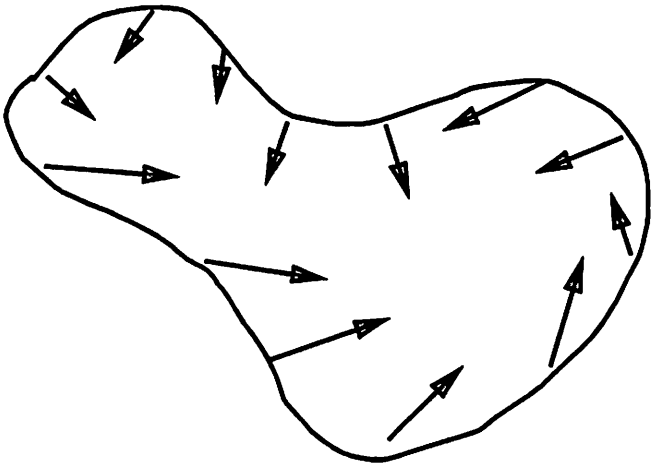

Figure 4.6: An ϵ-invariant set

4.5 Invariant Sets and Other Applications

In this section, we will use the results from Section 4.3 and 4.4 to compute the ϵ-invariant sets of differential inclusions. These are sets which remain invariant under the inclusion $\dot{x} \in f_\epsilon(x)$. We also show how to use the sample graph A^ϵ to decide various other properties of the differential inclusion $\dot{x} \in f(x)$.

4.5.1 Invariant Sets

Intuitively I is an invariant set provided f points "inwards" at the boundary. The idea is formalized using *contingent cones.*

Definition 4.5.1 *Let I be a closed set. The contingent cone to I at x is the set*

$$T_I(x) = \{v | \lim_{h \to 0} \frac{dist((x + hv), I)}{h} = 0\}$$

The result characterizing invariant sets under $\dot{x} \in f(x)$ is that I is invariant under f iff for $x \in I$, $f(x) \subset T_I(x)$ [6, 1].

Definition 4.5.2 *A set I is ϵ-invariant provided for all $x \in I$, $f_\epsilon(x) \subset T_I(x)$.*

Definition 4.5.2 states that the set I is ϵ-invariant provided it is invariant under ϵ-perturbations of f. Figure 4.6 shows an example of a ϵ-invariant set for some $\epsilon > 0$. Since f is not "tangential" to the boundary, the set remains invariant under small perturbations in f. Notice, we only need to check the condition of Definition 4.5.2 at the boundary of I, since $T_I(x) = \mathcal{R}^n$ in the interior of I. Any trajectory starting from an initial condition inside the invariant set remains inside the invariant set.

We next make a relationship between the invariant sets of differential inclusions and the infinite time reachability problem. To compute invariant set of $\dot{x} \in f(x)$ in R, we can compute $J = Reach\_f(R^c, [0, \infty))$ for $\dot{x} \in -f(x)$. Then J^c is the largest invariant set of $\dot{x} \in f(x)$ in R.

Lemma 4.5.1 *For a region R, the largest invariant set of $\dot{x} \in f(x)$ in R is $I_0 = (Reach\_f(R^c, [0, \infty)))^c$.*

Proof: $(Reach\_f(R^c, [0, \infty)))^c$ is the set of states in R which cannot reach R^c by following the inclusion $\dot{x} \in f(x)$. Hence, it is the largest invariant set in R. ∎

Similarly $I_\epsilon = (Reach\_f_\epsilon(R^c, [0, \infty)))^c$ is the largest ϵ-invariant set in R.

Theorem 4.5.1 *For a Lipschitz differential inclusion $\dot{x} \in f(x)$ on a region R, and $\epsilon > 0$, we can compute an invariant set I such that $I_\epsilon \subset I \subset I_0$.*

Proof: Consider the differential inclusion $\dot{x} \in -f(x)$. From Theorem 4.3.2 and Theorem 4.4.3, we can compute a set J such that $Reach\_f(X_0, [0, \infty)) \subset J \subset Reach\_f_\epsilon(X_0, [0, \infty))$ and J is invariant for $\dot{x} \in -f(x)$. Therefore $I = J^c$ is invariant for $\dot{x} \in f(x)$ and $I_\epsilon \subset I \subset I_0$. ∎

As a consequence of Theorem 4.4.3, the invariant set I in Theorem 4.5.1 can be computed in a finite number of steps.

4.5.2 Other Dynamical Properties

We consider various computational questions about limit cycles, attracting sets and stability for the differential inclusion $\dot{x} \in f(x)$ in a bounded region R. In each case, we ask whether the inclusion $\dot{x} \in f(x)$ has a certain property. To answer the

question, we work with the sample graph A^ϵ. In all cases, we obtain either a positive answer for the differential inclusion $\dot{x} \in f(x)$, or a negative answer for the inclusion $\dot{x} \in f_\epsilon(x)$. That is, the algorithm returns with one of the following answers: 1) Differential inclusion $\dot{x} \in f(x)$ has the desired property; or 2) Differential inclusion $\dot{x} \in f_\epsilon(x)$ does not have the property. The time complexity of the algorithm is same as the size of the graph A^ϵ — $O((\frac{k}{\epsilon})^n)$, where k is the lipschitz constant of the differential inclusion $\dot{x} \in f(x)$ and n is the dimension of the state space.

Limit Cycles

Consider the following problem: *Is it the case that the differential inclusion* $\dot{x} \in f(x)$ *does not have a limit cycle in region* $S \subset R$? To answer the question, restrict the graph A^ϵ to S, obtaining subgraph $(A^\epsilon)_S$. We look for the strongly connected sets in the subgraph. If there are no strongly connected sets in $(A^\epsilon)_S$, then Theorem 4.4.2 implies that $\dot{x} \in f(x)$ has no limit cycles in S. Alternatively if subgraph $(A^\epsilon)_S$ has a strongly connected set, then it follows that $\dot{x} \in f_\epsilon(x)$ has a limit cycle in S.

Attracting Sets

A set $B \subset R$ is an attracting set provided every trajectory starting from any initial condition in R enters B. Consider the problem: *Is* B *an attracting set for differential inclusion* $\dot{x} \in f(x)$? To answer the question, we look at the subgraph obtained from A^ϵ by deleting vertices in set B. If the subgraph does not contain a strongly connected set, then Theorem 4.4.2 implies that B is an attracting set for $\dot{x} \in f(x)$. On the other hand, if the subgraph contains a strongly connected set, then B is not attracting set for the inclusion $\dot{x} \in f_\epsilon(x)$.

Other Properties

Various other dynamical properties for differential inclusions can also be checked using the graph A^ϵ in a similar manner. For example, to determine whether an equilibrium point x is globally stable, we can ask whether $B_\delta(x)$ — the δ-ball centered at x — is an attracting set for some $\delta > 0$.

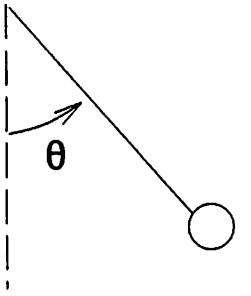

Figure 4.7: A Pendulum

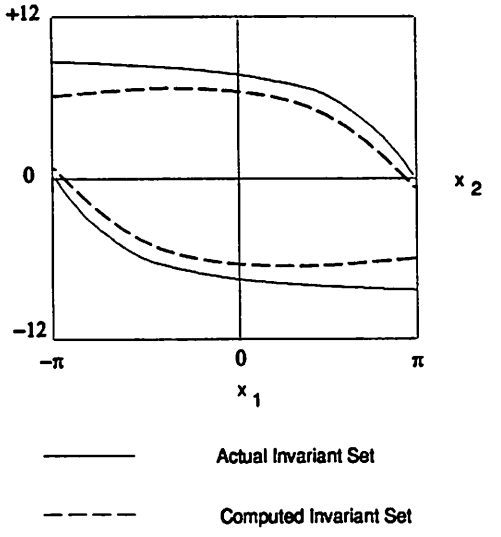

Figure 4.8: Invariant Set for the Pendulum

4.6 Computational Aspects and Examples

In this section we compute the invariant sets of some differential equations using the methods discussed in the previous sections. We also describe some techniques which can be used to increase the efficiency of these methods.

4.6.1 Pendulum Example

Figure 4.7 shows a pendulum moving in a vertical plane. The differential equation describing the pendulum is

$$\dot{x}_1 = x_2 \tag{4.3}$$

$$\dot{x}_2 = -g\sin(x_1) - cx_2$$

where $x_1 = \theta$, $x_2 = \dot{\theta}$, $g = 9.8$ is the acceleration of gravity and $c = 1$ is the frictional coefficient. We want to compute the largest invariant set contained in region $R = [-\pi, \pi] \times [-12, 12]$. The invariant set can be calculated exactly. It's boundary is given by $x_1 = -\pi$, $x_1 = \pi$, $U = \{x | x \in R \text{ and } \lim_{t\longrightarrow\infty} \phi(x,t) = (\pi, 0)\}$, and $L = \{x | x \in R \text{ and } \lim_{t\longrightarrow\infty} \phi(x,t) = (-\pi, 0)\}$ where $\phi(x,t)$ is the solution of Equation 4.3 with initial condition x. The actual invariant set is shown in Figure 4.8. The invariant set is an attracting set of the equilibrium point $(0,0)$ (see [16]).

We next use the graph method of Theorem 4.5.1 and Theorem 4.4.3 to compute the invariant set for Equation 4.3. A straightforward computation shows that $k = 11$ is a Lipschitz constant. We choose $\Delta = \frac{ln2}{k}$ to be the sample time (see Example 4.4.3). The graph is constructed on a grid of size 300×300 on R ($\beta = \frac{\pi}{150} = 0.021$) using the *Modified Sample Graph Construction.* We use Lemma 4.2.2 to obtain an approximation of $S_f(\langle g \rangle)$ where g is a grid point. The algorithm of Section 4.2 is then used to compute the reach set of the graph. The computed invariant set is shown in Figure 4.8. We note that the computed set is an invariant set, and it contains any ϵ-invariant set for $\epsilon \geq 2\beta(k + \frac{1}{\Delta})$ (i.e., $\epsilon \geq 1.12$).

As is to be expected, finer grids (smaller values of β) give better approximation of the actual invariant set, and coarse grids give a worse approximation. In particular, for grids with $\beta \geq 0.045$ we get no invariant set. From discussion in Example 4.4.3, it follows that there is no ϵ-invariant set in R for $\epsilon \geq 2.41$.

The size of the grid required to compute the invariant sets is also a function of the parameters c and g in Equation 4.3. For a lightly damped system (smaller c), we expect a larger grid size to be required. This is also observed in our computational experiments.

4.6.2 The Lorenz Equations

In this section, we study the Lorenz equations [16, 39]. The equations are

$$\dot{x} = \sigma(y - x) \tag{4.4}$$

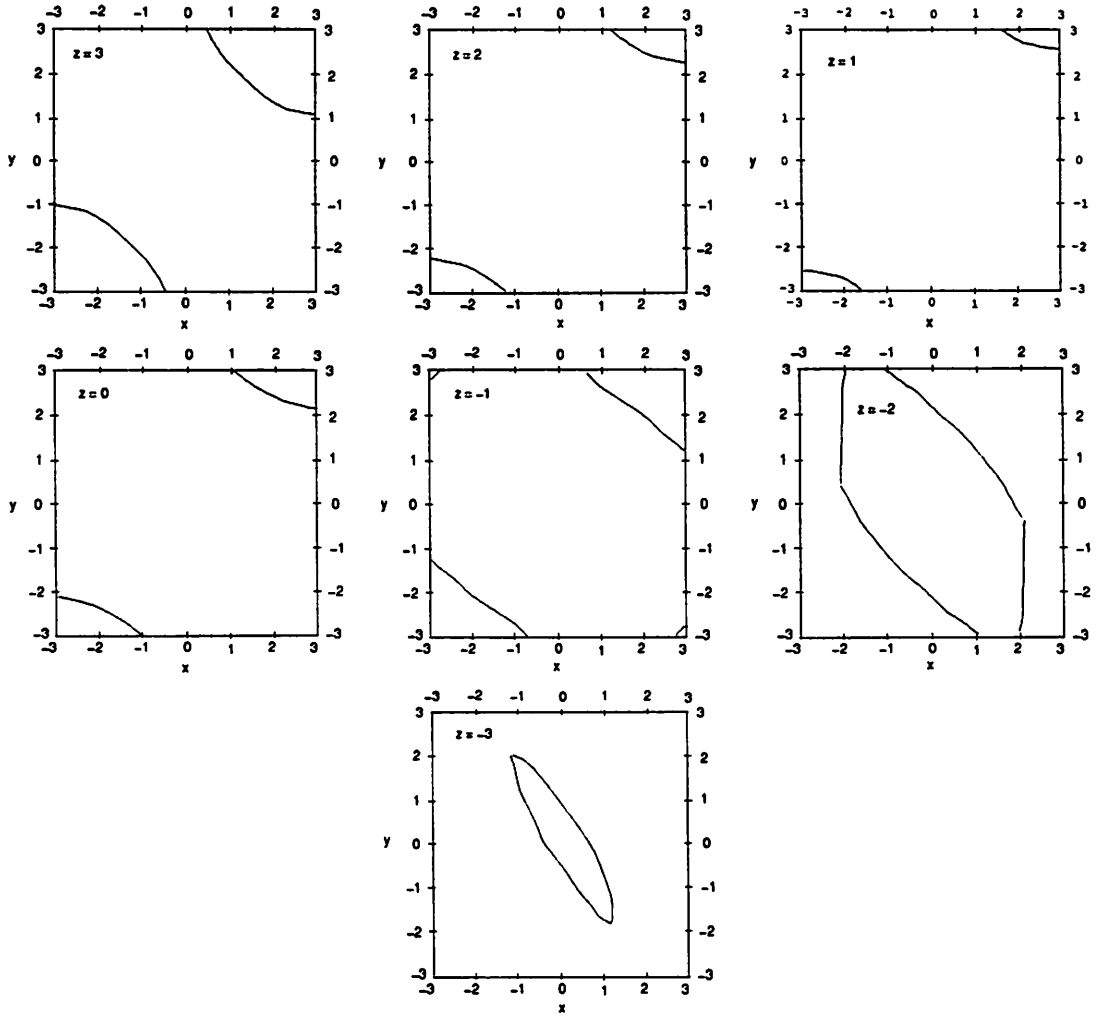

Figure 4.9: Computed Invariant Set for the Lorenz Equations

$$\dot{y} = \rho x - y - xz$$

$$\dot{z} = -\beta z + xy$$

We study the system for $\sigma = 1$, $\rho = 2$ and $\beta = 1$. The system has three fixed points (see [16, 39] for a detailed description of the dynamics of the system). The fixed point at the origin is a saddle point with one dimensional unstable manifold. The other two fixed points at (1,1,1) and (-1,-1,1) are stable. We compute the largest invariant set contained in the region $R = [-3, 3] \times [-3, 3] \times [-3, 3]$.

It can be shown that $k = 9$ is a Lipschitz constant. The sample time is $\Delta = \frac{ln2}{k}$

(see discussion in Example 4.4.3). A graph is constructed on a grid of size $90 \times 90 \times 90$. The algorithm of Section 4.2 is then used to compute the invariant set. In Figure 4.9, we show the cross sections of the invariant set for $z = k$, $k \in \{-3, -2, \ldots, 2, 3\}$.

4.6.3 Efficient Storage Methods

The main limitation of the grid method is the amount of memory required to store the grid points. The main limitation of the polyhedral method is that the iteration in the *Reach Set Computation* may never terminate. It may be possible to combine the desirable features of the two methods to obtain better efficiency. Rather than storing the grid points explicitly, a set of grid points can be stored as a polyhedron using linear inequalities. Similarly the termination problem in the *Reach Set Computation* for the polyhedral method may be addressed by using some features of the grid method.

4.7 Conclucion

We presented a method to compute $Reach_f(X_0, t)$ — the reach set of the differential inclusion $\dot{x} \in f(x)$ at time t. We also presented a method to compute the ϵ-invariant sets of the inclusion $\dot{x} \in f(x)$. We do this by relating the differential inclusion $\dot{x} \in f(x)$ to an ϵ-perturbation $\dot{x} \in f_\epsilon(x)$. This closely resembles concepts from perturbation theory, and structural stability in dynamical systems. In our future work, we will look into this relationship further. In particular, we would like to show that for sufficiently small ϵ, the presence of a property in $\dot{x} \in f_\epsilon(x)$ also implies the presence of that property for $\dot{x} \in f(x)$ for some class of properties.

Computing the invariant sets and reach sets is an important problem. It is finding increasing use in the study of hybrid systems (see [31, 33]). An important problem is to find techniques to make the algorithms and methods presented in this chapter more efficient in terms of space and time usage.

Chapter 5

Shapley's Game and Church's Problem

5.1 Introduction

In this chapter, we study graph games played by two players. The games starts from start position p_0 with Player 1 moving to position p_1 along an edge. Then Player 2 moves to position p_2, followed by Player 1 making a move, and so on. In this way, an infinite play $p = p_0 p_1 p_2 \ldots$ is constructed. At the "end," Player 2 pays to Player 1 the amount $P(p)$ where P is a payoff function. The games are perfect information zero-sum games. So Player 2 tries to minimize the payoff to Player 1, and Player 1 tries to maximize it.

A strategy for a player is a rule which tells him how to play. A game has a value v when Player 1 has a strategy which assures him a payoff of at least v and Player 2 has a strategy which guarantees her a loss no greater than v. The strategy which assures a payoff of v for Player 1 is his optimal strategy, and the strategy which guarantees for Player 2 a loss no greater than v is her optimal strategy. By "solving a game," we mean determining the value of the game, and the optimal strategies for the players. We study two classes of games: the payoff games and games on ω-automata.

Payoff games are a special case of stochastic games introduced by Shapley [38] and extensively studied in the operations research community since then [36]. In the

payoff game, each position w has some reward $r(w)$. The players move from position to position, creating the play $p = p_0 p_1 p_2 \ldots$. The payoff is $P(p) = \sum_k r(p_k)$. Player 1 chooses his moves to maximize the payoff; Player 2 chooses hers to minimize it.

An ω-automaton is a finite state automaton which accepts infinite sequences [40, 23]. In a game on ω-automaton, the objective of Player 1 is to create a play p which is accepted by the automaton; Player 2 tries to create a play which is rejected. Solving a game on an ω-automaton is sometimes referred to as Church's problem who posed the question as a synthesis problem for digital circuits [8]. Games on ω-automata have been extensively studied in the logic and computer science communities [35, 40, 23].

In this chapter, we relate games on ω-automata with the payoff games. We show that a game on an ω-automaton with the chain acceptance condition can be solved as a payoff game. The chain acceptance condition can express any ω-regular language. As a result, any game on an ω-automaton can be solved as a payoff game. It is known that solving the model checking problem for propositional μ-calculus is polynomially equivalent to solving the chain game [13]. Hence, we get a new method for model checking μ-calculus using algorithms for solving payoff games.

In Section 5.2, we introduce games played on graphs. In Section 5.3, we discuss games on ω-automata. We review results on ω-automata games and show that such games do not have a value when restricted to positional strategies. In Section 5.4, we begin discussing payoff games. We study both finite and infinite games. For the infinite game, we study the discounted payoff game and the mean payoff game. We prove that both games have a value and optimal positional strategies. We then introduce the successive approximation and the policy iteration algorithms for solving payoff games. In Section 5.5, we relate the chain games with the payoff games. We show that the results for payoff games also provide answers to classical problems such as Church's solvability and synthesis problem. As a result of the relationship between chain games and payoff games, we can use the policy iteration algorithm to model check μ-calculus formulae.

5.2 Games on Graphs

A game graph is a directed bipartite graph $G = (V, E)$. The positions in the game are the vertices $V = V_1 \cup V_2$ where V_1 and V_2 are disjoint. The edges are $E = E_1 \cup E_2$ where $E_1 \subset V_1 \times V_2$ and $E_2 \subset V_2 \times V_1$. Without loss of generality, we assume $V = \{1, \ldots, 2n\}$. The index set $I = \{1, \ldots, n\}$, and the position set $V_1 = \{2i | i \in I\}$ and $V_2 = \{2i - 1 | i \in I\}$. Furthermore, there are exactly two outgoing edges from each vertex. Hence, we may interpret the set of edges as functions $e_0 : V \longrightarrow V$ and $e_1 : V \longrightarrow V$ where the edges out of vertex v are $(v, e_0(v))$ and $(v, e_1(v))$. The neighbors of a vertex v are $N(v) = \{e_0(v), e_1(v)\}$.

The game is played between two players: Player 1 and Player 2. From a position $v \in V_1$, Player 1 can make a move to $e_0(v)$ or $e_1(v)$. Similarly, Player 2 moves from $w \in V_2$ to $e_0(w)$ or $e_1(w)$. For convenience, we will refer to Player 1 as "him", and Player 2 as "her".

The game begins from the start position $p_0 \in V_1$ with Player 1 making a move. Then it is Player 2's turn to move, and so on. The resulting play of the game is $p = p_0 p_1 p_2 \ldots$ where Player 1 moved from position p_{2i} to p_{2i+1}, and Player 2 moved from position p_{2i+1} to p_{2i+2}. The payoff is a function $P : V^\omega \longrightarrow \mathcal{R}$. The games we are interested in are perfect information zero-sum games. When the play is p, Player 2 pays to Player 1 the amount $P(p)$ (when $P(p)$ is negative, Player 1 pays to Player 2). Player 1 tries to maximize the payoff, and Player 2 tries to minimize it.

A game is a triple $\langle G, p_0, P \rangle$ where $G = (V, E)$ is a game graph, $p_0 \in V$ is the position from which the game starts, and P is the payoff function.

Example 5.2.1 *Consider the game graph shown in Figure 5.1. The positions represented with circles are V_1. Those represented with squares are V_2. There is an arrow pointing to the start position. When the game position is a circle, Player 1 moves; when the game is at a square, Player 2 moves. The edges out of each position represent the moves the players can make. So from position D, Player 2 can make a move to either position C, or to position B. For a play p, let us define the payoff function to be $P(p) = -1$ when $p_i = E$ or $p_i = F$ for some i. Otherwise $P(p) = 1$. So, Player*

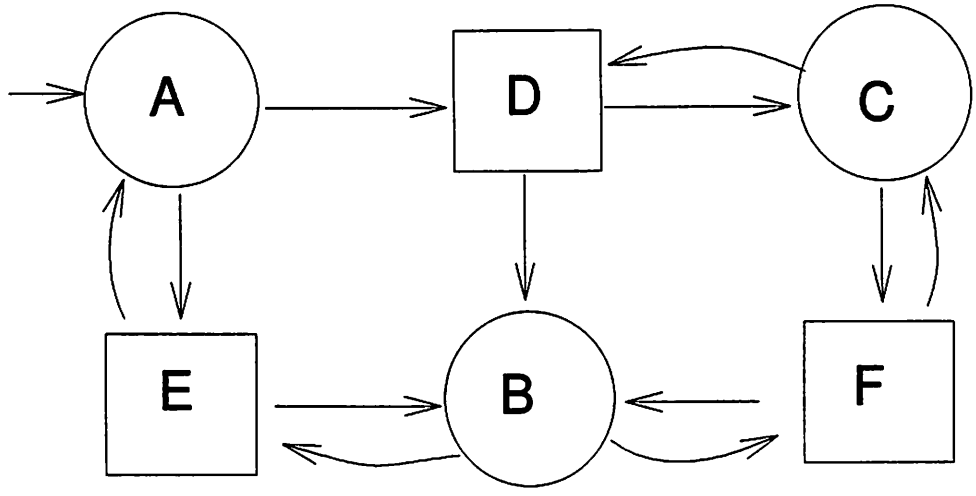

Figure 5.1: An Example Game

2 gets paid 1 dollar provided the play p "hits" the vertices E or F. On the other hand, if those vertices are not hit, Player 2 has to pay 1 dollar to Player 1.

We will say Player 1 wins in the play p provided $P(p) > 0$; otherwise Player 2 wins. We leave it to the reader to find the winner in the game of Figure 5.1.

5.2.1 Strategies for Playing

A strategy for a player is a rule or a set of rules which tells the player how to play. We first discuss history based strategies. The history of the play p at step j is $h_j = p_0 p_1 \dots p_j$. A history based strategy for Player 1 is the set of functions $\tau = \{\tau_{2i}\}$ from the histories of the play to its next move. At step $2i$ when h_{2i} is the history at step $2i$, Player 1 makes a move to position $\tau_{2i}(h_{2i})$. Similarly, a history based strategy for Player 2 is the set of functions $\sigma = \{\sigma_{2i+1}\}$. Let us denote by HD the set of all history based strategies.

In many games, it is possible to play with a simpler class of strategies: the positional strategies. A positional strategy for Player 1 is a function $\tau : V_1 \longrightarrow V_2$ where $(v, \tau(v)) \in E_1$. When in position v, Player 1 always moves to $\tau(v)$, independent of the history of the play. A positional strategy for Player 2 is a function $\sigma : V_2 \longrightarrow V_1$. Positional strategies are also called deterministic Markov strategies. Let us denote by

MD the set of positional strategies. Corresponding to a positional strategy τ is the history based strategy $\{\tau_{2i}\}$ where $\tau_{2i}(h_{2i}) = \tau(p_{2i})$. When convenient, we will also think of positional strategy τ as the set of edges $\{(v, \tau(v))\}$.

5.2.2 Value of the Game

Denote by A the strategy space for Player 1 and by B the strategy space for Player 2. Fixing a strategy $\tau \in A$ for Player 1 and $\sigma \in B$ for Player 2 leads to the play $p = p_0 p_1 \ldots$ where $p_{2i+1} = \tau_{2i}(p_{2i})$ and $p_{2i+2} = \sigma_{2i+1}(p_{2i+1})$.

Definition 5.2.1 *The strategy payoff function is* $\psi : A \times B \longrightarrow \mathcal{R}$ *where* $\psi(\tau, \sigma) = P(p)$. *The play* p *is obtained by fixing the strategy of Player 1 to* τ *and the strategy of Player 2 to* σ.

Definition 5.2.2 *A game is said to have a value in strategy space* (A, B) *provided*

$$\sup_{\tau} \inf_{\sigma} \psi(\tau, \sigma) = \inf_{\sigma} \sup_{\tau} \psi(\tau, \sigma), \tag{5.1}$$

where the strategies for Player 1 are selected from A, and the strategies for Player 2 are selected from B. The value of Equation 5.1 is the value for the game in strategy space (A, B).

The value for the game in strategy space *(HD,HD)* is the *value of the game.* When the value of the game is v, and $v > 0$, we say Player 1 wins the game. Otherwise, Player 2 wins.

It will also be useful to study the class of games $\mathcal{G} = \{\mathcal{G}_j\} = \langle G, P \rangle$ where $\mathcal{G}_j = \langle G, j, P \rangle$ is the game played on graph G starting from position j. The value of $\mathcal{G}$ is a vector u where $u(j)$ is the value of game $\mathcal{G}_j$.

5.2.3 Complete Strategy Spaces and 1-Player Games

Definition 5.2.3 *For Player 1, the strategy space MD is complete for Player 1 provided the value for the game in (MD,HD) is the value of the game. Completeness for Player 2 is defined similarly.*

A player needs to search only among positional strategies when the strategy space MD is complete.

Definition 5.2.4 *The response function for Player 2 is* $r_2 : A \longrightarrow B$ *where*

$$r_2(\tau) = \arg\min_{\sigma} \psi(\tau, \sigma).$$

We note that $r_2(\tau)$ is a set of strategies — each strategy in $r_2(\tau)$ is the optimal strategy for Player 2 in response to Player 1 playing with strategy τ. The response function for Player 1, r_1, is defined similarly. Strategy $\tau \in A$ and $\sigma \in B$ are said to be optimal in (A, B) provided $\tau \in r_1(\sigma)$ and $\sigma \in r_2(\tau)$.

A 1-Player game is obtained from game $\mathcal{G}$ by fixing the strategy of one of the players to a positional strategy.

Definition 5.2.5 *Fixing positional strategy* τ *for Player 1, define the 1-Player game* $\mathcal{G}_\tau = (G_\tau, P)$ *where* $G_\tau = (V, E_2 \cup \tau)$.

In game $\mathcal{G}_\tau$, Player 2's objective is to minimize the payoff when Player 1 has fixed his strategy to τ. The optimal strategy for Player 2 in game $\mathcal{G}_\tau$ is $\sigma \in r_2(\tau)$.

Definition 5.2.6 *We say MD is complete for Player 2 in* $\mathcal{G}_\tau$ *provided* $r_2(\tau)$ *contains a positional strategy.*

We can similarly define 1-Player games and completeness for Player 1.

Lemma 5.2.1 *Suppose* τ *and* σ *are optimal in* (MD, MD) *for game* $\mathcal{G}$*, and* MD *is complete for both players in 1-Player games. Then* τ *and* σ *are also optimal in* (HD, HD).

5.3 Games on ω-Automata, Church's Problem and μ-Calculus

ω-automata are finite automata which accept infinite sequences. The sequences which are accepted form the language of the ω-automaton [40, 23]. A game on an ω-automaton is played between Player 1 and Player 2. The two players together create

a sequence $p = p_0 p_1 p_2 \ldots$. Player 1 wins provided the sequence is in the language of the ω-automaton; otherwise, Player 2 wins. In Section 5.3.1, we discuss ω-automata. In Section 5.3.2, we discuss games on ω-automata. In Section 5.3.3, we discuss the logic propositional μ-calculus.

5.3.1 ω-Automata

A deterministic ω-automaton over alphabet Σ is $\mathcal{A} = \langle T, \phi \rangle$, where T is the transition structure, and ϕ is the acceptance condition. The transition structure is $T = (Q, q_0, \Sigma, \delta)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, and $\delta : Q \times \Sigma \longrightarrow Q$ is the transition function.

Definition 5.3.1 *A word $\sigma \in \Sigma^\omega$ has the run $r_\sigma \in Q^\omega$ where $r_\sigma(0) = q_0$ and $\delta(r_\sigma(i), \sigma(i)) = r_\sigma(i+1)$.*

The infinity set of the run r_σ, denoted $\inf(r_\sigma)$, is the set of states that are visited infinitely often in r_σ. The acceptance condition ϕ is a boolean formula, where the boolean variables are the states $Q = \{q_1, \ldots, q_m\}$.

Definition 5.3.2 *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 the boolean variable $q_i \in Q$ is determined by the run r_σ. For $C \subset Q$, define the boolean assignment $q_i = 1$ provided $q_i \in C$, otherwise $q_i = 0$. Let $\phi[C]$ denote the truth value of ϕ under this assignment.

Definition 5.3.3 *The language generated by the ω-automaton $\mathcal{A} = \langle T, \phi \rangle$ is $\mathcal{L}(\mathcal{A}) = \{\sigma | \sigma \in \Sigma^\omega$ and $\phi[\inf(r_\sigma)] = 1 \}$.*

Definition 5.3.4 *We define various types of boolean formulae which are used for acceptance criterion:*
1) A disjunctive formula (DF) (Buchi formula) is a disjunction of boolean variables, i.e., for $F = \{f_1, \ldots, f_k\} \subset Q$, $\phi = f_1 \vee \ldots \vee f_k$.

2) A Rabin formula is $\phi = \bigvee_{i=1}^{n}(L_i \wedge \neg(\bar{U}_i))$ *where* $L_i, U_i, 1 \leq i \leq n$ *are DF.*
3) A Streett formula is $\phi = \bigwedge_{i=1}^{n}(L_i \vee \neg(\bar{U}_i))$ *where* $L_i, U_i, 1 \leq i \leq n$ *are DF.*
4) Given DF $E_i, F_i, i = 1, \ldots, n$ *where* $E_n \subset F_n \subset E_{n-1} \subset F_{n-1} \subset \ldots E_1 \subset F_1$, *the chain formula is* $\phi = \bigvee_{i=1}^{n}(F_i \wedge \neg E_i) \equiv \bigwedge_{i=1}^{n+1}(\neg E_{i-1} \vee F_i)$ *where* $E_0 = true$ *and* $F_{n+1} = false$.

The complement of a Rabin formula is a Streett formula, and vice versa. The chain formula can be expressed either as a Rabin formula, or as a Streett formula.

The chain acceptance condition is also sometimes written as the parity acceptance condition (B_i, G_i) where $B_i = E_i \setminus F_{i+1}$ and $G_i = F_i \setminus E_i$ (see [23, 13]). The sets B_i and G_i are disjoint. A run r_σ is accepting provided for some i, $\inf(r_\sigma)$ "touches" G_i but not B_j for $j \geq i$.

The Rabin and Streett acceptance conditions are also referred to as a set of pairs (L_i, U_i) of subsets of states. A run r_σ in a DRA is accepting if for some pair, $\inf(r_\sigma)$ "touches" L_i and is contained in U_i.

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.

The ω-automaton $\mathcal{A} = \langle T, \phi \rangle$, where ϕ is a Buchi, Rabin, Streett, or Chain formula is called a Buchi, Rabin, Streett, and Chain Automaton respectively. We will need the following two results from the theory of ω-automata.

Theorem 5.3.1 *Determining whether the language of* $\mathcal{A} = \langle T, \phi \rangle$ *is empty for Buchi, Rabin, Streett, and Chain Automata is in polynomial time.*

Theorem 5.3.2 *Given a Rabin Automaton* $\mathcal{A} = \langle T, \phi \rangle$ *with* n *states and* h *pairs, there is a Chain Automata* $\mathcal{B}$ *with* nh^k *states and* k *pairs such that* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$. *The index* k *is the Rabin Index of the language — the minimum number of pairs in a Rabin automaton required to realize the language.*

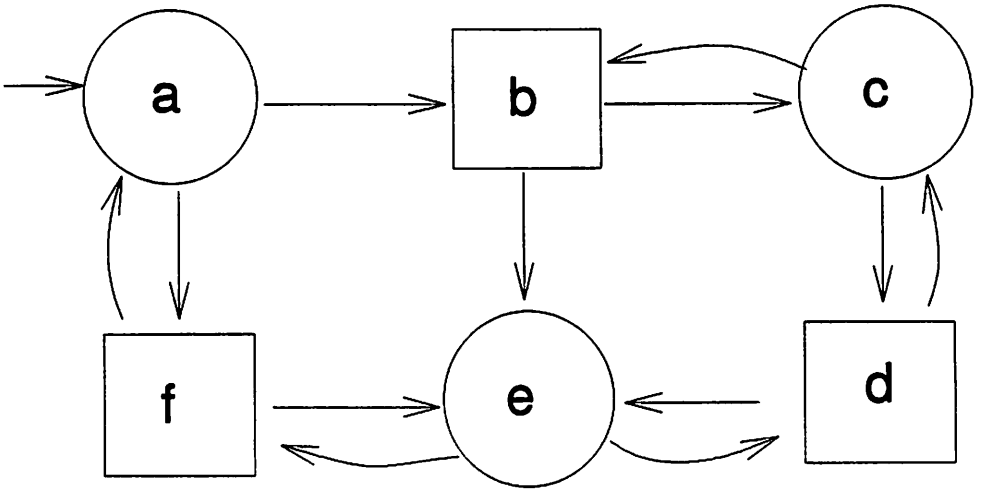

Figure 5.2: An Game on Buchi Automaton

For a more complete discussion of ω-automata and a proof of Theorem 5.3.1, see [40]. Theorem 5.3.2 is the main result of [23].

5.3.2 Games on ω-Automata

A game on an ω-automaton is $\mathcal{G} = (G, P_\phi)$ where $G = (V, E)$ is a game graph and ϕ is an acceptance condition. The acceptance condition ϕ is a boolean formula with boolean variables $V = \{v_1, \ldots, v_{2n}\}$. For a Buchi, Rabin, Streett, or Chain formula ϕ, we say the game is a Buchi, Rabin, Streett, and Chain game respectively.

Definition 5.3.5 *The payoff function for the game is* $P_\phi : V^\omega \longrightarrow \{0, 1\}$, *where for a play* p, $P_\phi(p) = \phi[\inf(p)]$.

For a play p, Player 1 is the the winner provided $\phi[\inf(p)] = 1$, and Player 2 is the winner when $\phi[\inf(p)] = 0$. Player 1 tries to maximize the payoff, and Player 2 tries to minimize it.

Example 5.3.1 *Consider the formula* $\phi = v_1 \vee \ldots \vee v_k$ *where* $F = \{v_1, \ldots, v_k\}$ *are called the final states. For Player 1 to win, the play must visit some state in* F *infinitely often. On the other hand, Player 2 tries to limit the number of times a state in* F *is visited. For the game graph in Figure 5.2,* $F = \{b, e\}$ *and the boolean*

formula $\phi = b \vee e$. To win, Player 1 must use a strategy, such that any resulting play visits position b or position e infinitely often. Consider the positional strategy τ for Player 1 where $\tau(a) = b$, $\tau(c) = b$ and $\tau(e) = f$. The reader can check that starting from any position, this is a winning strategy for Player 1 since any resulting play visits a state in F infinitely often.

Games on ω-automata are closely related to synthesis problems. Church in [8] poses two problems about digital circuits and logic: the decision problem and the synthesis problem. The decision problem is the verification question: does the designed circuit meet its specification? The synthesis problem is to automatically synthesize a circuit to meet a specification. He gives various cases of the synthesis and decision problem which had been solved, but points out that the synthesis question for ω-automata was unresolved.

In the synthesis problem, the specification is a game which is played by two players: the "input" (also sometimes called the "disturbance") and the "controller". The idea is that for every input to the circuit, the controller must respond so that the specification is never violated. When the controller can win this game, we say the specification is synthesizable. The winning strategy for the controller is implemented as the circuit. This circuit then meets its specification.

Solving a game on ω-automaton is sometimes called Church's problem. In [40], Church's problem is presented as the following two questions about games on ω-automata:

1. Solvability Problem — Is there an algorithm to determine the winner?

2. Synthesis Problem — Is there a finite-state strategy for the winner?

The problem was resolved by Buchi, Landweber and Rabin (see [35]).

Theorem 5.3.3 *Consider the game $\mathcal{G} = (G, P_\phi)$ where ϕ is a boolean formula. Then*

1. the game $\mathcal{G}$ has a value (i.e., the game has a winner),

2. there is an algorithm to determine the winner,

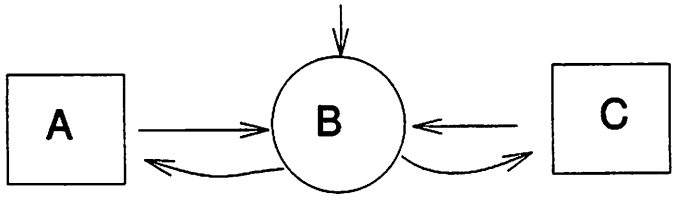

Figure 5.3: The value in (MD,MD) and (HD,HD) are not the same

3. the winner can implement its strategy with a finite amount of memory.

The following is the result of [11].

Theorem 5.3.4 *Strategy space MD is complete for Player 1 in Rabin Games.*

Hence, Player 1 can restrict himself to playing with positional strategies in Rabin Games. Since a Chain formula is a special case of a Rabin formula and a Streett formula, and the complement of a Streett formula is a Rabin formula, we get the following result.

Lemma 5.3.1 *The strategy space MD is complete for both players in Chain Games.*

At this time, one may ask the following questions: Are there games in which a player needs to remember the history? Do games on ω-automata have a value in the strategy space *(MD,MD)*? Does the value in *(MD,MD)* coincide with the value in *(HD,HD)* ?

We first show with an example that history based strategies are required, and that the value in *(MD,MD)* need not coincide with the value in *(HD,HD)*.

Example 5.3.2 *Consider the Streett game in Figure 5.3 where the acceptance condition is $A \wedge C$. We use the convention that when both edges out of a position are going to the same position, we only draw one edge. To win, Player 1 has to visit A and C infinitely often. Consider the positional strategy where Player 2 always plays to B from A and C. There is no positional strategy for Player 1 which can beat this. Hence the value of the game in (MD,MD) is* 0. *But using a history based strategy Player 1 can win. He just alternately visits A and C. Therefore, the value of the game in (MD,MD) and (HD,HD) need not coincide.*

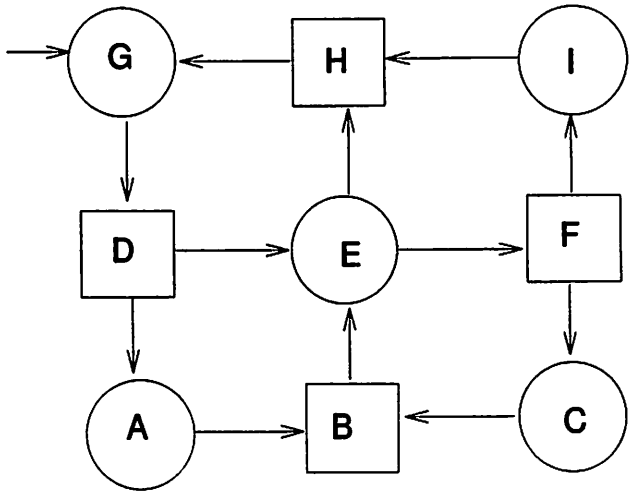

Figure 5.4: The game does not have a value for (MD,MD)

In the next example, we show that a game need not a have a value for the strategy space *(MD,MD)* .

Example 5.3.3 *Consider the Rabin game shown in Figure 5.4 where the acceptance condition is* $\phi = C \vee (I \wedge \neg A) \vee (A \wedge \neg I)$. *Player 1 wins by visiting infinitely often (i.o.) position C, or visiting i.o. position I but not A, or by visiting i.o. position A but not I. We will leave it to the reader to check that any positional strategy of Player 1 can be beaten by Player 2 playing a positional strategy. And any positional strategy of Player 2 can be beaten by a positional strategy of Player 1. Hence the game does not have a value for the strategy space (MD,MD). But the game is won by Player 2 using the following history based strategy: Player 2 always plays to position I from position F. At position D, she looks at what Player 1 did during his last move at E: when Player 1 moved to position F, Player 2 moves from D to A; if Player 1 moved to position H, then Player 2 moves from D to E. The reader can check that this is a winning strategy. We can now see the significance of the first part of Theorem 5.3.3 which assures us that a game on an* ω*-automaton always has a value in strategy space* (HD,HD).

Given an ω-automaton of one type, we can convert it to another kind such that the languages of the two are the same. The same construction can be applied to convert

one type of game into another type such that both games have the same winner, and the winner can use the "same" strategy to win both games.

Let us clarify the meaning of the "same" strategy. The history upto step j is $h_j = p_0 p_1 p_2 \ldots p_j$. A strategy for Player 1 is a set of functions $\tau = \{\tau_{2i}\}$ where at step $2i$, Player 1 moves to the position $\tau_{2i}(h_{2i})$. Corresponding to h_j is the string $\tilde{h}_j = b_0 b_1 \ldots b_{j-1}$ where $b_i \in \Sigma = \{0,1\}$ and $p_{i+1} = e_{b_i}(p_i)$. We may view a strategy as functions $\tilde{\tau} = \{\tilde{\tau}_{2i}\}$ from strings in Σ^* to Σ. That is $w = \tilde{\tau}_{2i}(\tilde{h}_{2i}) \in \Sigma$ and at step $2i$, Player 1 moves to $e_w(p_{2i})$. Given a strategy τ and a game graph G, strategy $\tilde{\tau}$ is uniquely defined, and vice versa. Furthermore $\tilde{\tau}$ does not depend on the graph G. Strategies for Player 2 which are independent of the graph G could be defined similarly. When the game is converted into another type of game, the winner can use same strategy $\tilde{\tau}$ to win both games.

Lemma 5.3.2 *Given a Rabin Game $\mathcal{G} = (G, P_\phi)$ with n positions and h pairs, there is a Chain Game $\mathcal{C} = (C, P_\psi)$ with nh^k positions and k pairs where $k \leq h$, such that the same player wins both games, and can do so by using the same strategy. The index k is the Rabin Index of the Game language.*

Proof: Using Theorem 5.3.2. ∎

But a chain game can be played using positional strategies. Hence we obtain an upper bound on the amount of memory that Player 2 (Player 1) requires to play a Rabin Game (Streett Game).

Theorem 5.3.5 *In a Rabin Game (Streett Game) with n positions and h pairs, the amount of memory required by Player 2 (Player 1) to play is at most nh^k where k is the Rabin Index of the game language.*

1-Player ω-Automata Games

Definition 5.3.6 *Corresponding to the 1-Player game $\mathcal{G}_\tau = (G_\tau, P_\phi)$ with $G = (V, E_2 \cup \tau)$, define the ω-automaton $\mathcal{A} = \langle T, \phi \rangle$ with alphabet $\Sigma = \{0,1\}$ where $T = \langle V, q_0, \delta, \Sigma \rangle$ and for $i \in I$, $\delta(2i-1, 0) = e_0(2i)$, $\delta(2i-1, 1) = e_1(2i)$ and $\delta(2i, 0) = \delta(2i, 1) = \tau(2i)$.*

The ω-automaton $\mathcal{A}$ is obtained by labelling the edges of transition structure $\mathcal{G}_\tau$ with an alphabet. A player can win the game $\mathcal{G}_\tau$ provided he produces a play which satisfies acceptance condition ϕ. But this is exactly the emptiness problem for the ω-automaton $\mathcal{A}$. Hence, a player wins the 1-Player game $\mathcal{G}_\tau$ iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$.

Lemma 5.3.3 *Determining if a player wins the 1-Player Buchi, Rabin, Streett or Parity game is in polynomial time.*

Proof: From Theorem 5.3.1, checking emptiness of the corresponding ω-automaton is in polynomial time. ∎

We also know that the space *MD* is complete for Player 1 in Rabin games, and for both players in Chain Games. From Lemma 5.3.3, it follows that determining whether Player 1 wins a Rabin Game is in NP, a Streett Game in Co-NP, and a Chain Game in $NP \cap Co-NP$. In [12], it is shown that determining whether Player 1 wins a Rabin (Streett) game is NP-hard (Co-NP hard).

5.3.3 Propositional μ-Calculus

Propositional μ-calculus is a propositional modal logic with a least fixed point operator μ [22, 13].

Syntax

The logic has atomic propositions $A = p, q, \ldots$, and propositional variables $x, y, \ldots$. The set of formulas are defined inductively:

1. atomic propositions $p, q, \ldots$,
2. propositional variables $x, y, \ldots$,
3. $f \vee g$, and $\neg f$ where f, g are formulae,
4. $\langle a \rangle f$ where f is a formula,
5. $\mu x.f$ where formula f is positive in variable x.

Scope, occurence of free and bound variables, and closed formulas are defined similar to first-order logic.

Semantics

The model for a μ-calculus formula is a Kripke structure $K = (S, E, L)$ where S is a set of states, $E \subset S \times S$, and $L : S \longrightarrow 2^A$.

A valuation ρ is map which assigns sets of states to free variables. It assigns ρ_i to variable x_i. We will write $f^K(\rho)$ to denote formula f interpreted with valuation ρ. When f does not contain free variables, its interpretation does not depend on ρ, and we write it as f^K. We define $f^K(\rho)$ inductively as follows:

1. $x_i^K(\rho) = \rho_i$
2. $q^K(\rho) = \{s | q \in L(s)\}$
3. $(f \vee g)^K(\rho) = f^K(\rho) \cup g^K(\rho)$
4. $(\neg f)^K(\rho) = S \setminus f^K(\rho)$
5. $(\langle a \rangle f)^K(\rho) = \{s | (s, s') \in E \text{ and } s' \in f^K(\rho)\}$
6. $(\mu x_i.f)^K(\rho) = \bigcap\{X | X = f^K(\rho') \text{ where } \rho'_i = X \text{ and } \rho'_j = \rho_j \text{ for } j \neq i\}$.

We also write $[a]f = \neg\langle a \rangle \neg f$. The greatest fixed point ν is defined as $\nu x.f = \neg\mu x.\neg f$.

The least fixed point can be computed by starting from the emptyset, and iterating until a fixed point is reached. Similarly, the greatest fixed point is computed by starting from the set containing all states and iterating [22]. In this way, the set f^K can be computed inductively.

Given a Kripke structure $K = (S, E, L)$, a state $s \in S$, and a μ-calculus formula f, the *model checking problem* is to determine whether $s \in f^K$.

Chain Games and μ-Calculus

The following two theorems of [13] show that the model checking problem for μ-calculus and solving Chain games are essentially equivalent problems.

Theorem 5.3.6 *Given a Kripke structure $K = (S, E, L)$, state $s_0 \in S$, and μ-calculus formula f, there is a Chain Game $\mathcal{G}$ with $O((|S| + |E|)|f|)$ positions such that Player 1 wins the chain game $\mathcal{G}$ iff $s_0 \in f^K$.*

Theorem 5.3.7 *Given a Chain Game $\mathcal{G} = \langle G, p_0, P_\phi \rangle$, where $G = (V, E)$, there is a Kripke structure K of size $O(|V|)$ and a μ-calculus formula f of size $O(|\phi|)$ such that $p_0 \in f^K$ iff Player 1 wins game $\mathcal{G}$.*

Since we gave an inductive algorithm for model checking μ-calculus, we also obtain an algorithm for solving Chain games. The complexity of this algorithm is $O((|V|k)^k)$ where $|V|$ is the number of positions in the chain game, and k is the number of pairs. In Section 5.5, we will compare this algorithm with a new algorithm which we present in Section 5.4.

5.4 Shapley's Game

In this section, we discuss one of the simplest dynamical games. The game is played on the game graph $G = (V, E)$. There is a reward $r(v)$ at each position v. When the play in the game is $p = p_0 p_1 \ldots p_N$, Player 2 pays to Player 1 the amount $\Sigma_{i=0}^{N} r(p_i)$. Player 1 tries to maximize his payoff, and Player 2 tries to minimize it.

This class of games was originally studied by Shapley under a more general setting. Shapley in [38] introduced the class of stochastic games. A stochastic game is a zero-sum game played between two players: Player 1 who is trying to maximize his payoff, and Player 2 who wants to minimize it. The game has a finite number of positions. In each position, the players choose from a finite number of actions. When the game is in positon v, both players simultaneously choose their actions (say actions a and b), then Player 1 gets a reward r_v^{ab} and the game moves to the position w with probability

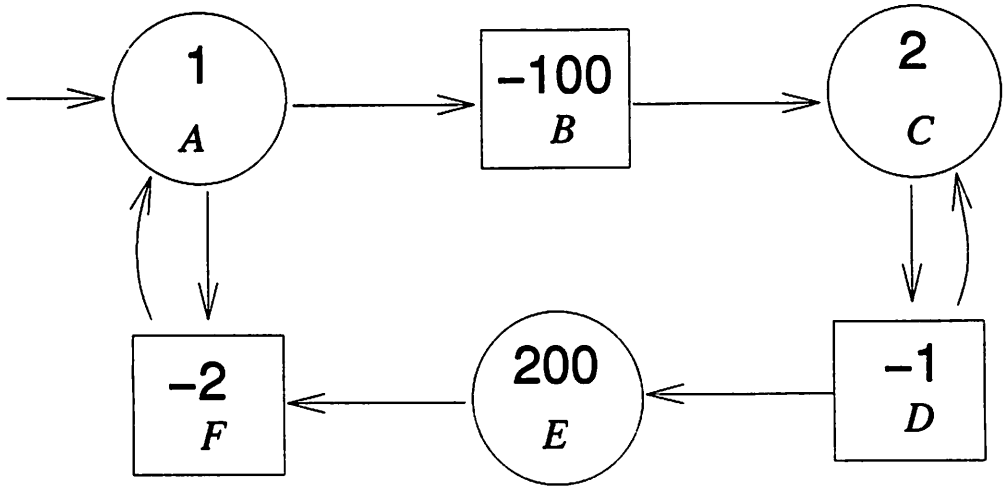

Figure 5.5: A Payoff Game

P_{vw}^{ab}. Both players again choose actions at position w and the game moves on to a new position, and so on.

Shapley studied stochastic games with the discounted payoff $P_\beta(p) = \sum_{i=0}^{\infty} \beta^i r_{p_i}^{a_i b_i}$, where $p = p_0 p_1 \ldots$ is the play of the game and a_i, b_i are actions of the players at step i and $0 < \beta < 1$. He showed that such games have a value, and both players have optimal stationary strategies [38].

We study a subclass of stochastic games — called *Payoff Games* — played on the game graph G. From positions $v \in V_1$, Player 1 chooses an action and moves to $e_0(v)$ or $e_1(v)$. From position $w \in V_2$, Player 2 moves to $e_0(w)$ or $e_1(w)$. The reward function $r : V \longrightarrow \mathcal{Z}$ is independent of actions. At a position v, Player 1 gets the reward $r(v)$. We study both finite and infinite games. Figure 5.5 shows an example of a payoff game.

In a finite N-step game the play is $p = p_0 p_1 \ldots p_N$. The payoff is

$$P_N(p) = \sum_{k=0}^{N-1} \beta^k r(p_k) + \beta^N f(p_N),$$

where f is the terminal reward function, and β is a discount factor.

For the infinite game, the play is $p = p_0 p_1 p_2 \ldots$. Two payoff functions are usually

studied. For a play $p = p_0 p_1 \ldots$, the discounted payoff is

$$P_\beta(p) = \sum_{k=0}^{\infty} \beta^k r(p_k)$$

where the discount factor $\beta \in (0,1)$. The average payoff is

$$\bar{P}(p) = \lim_{N\to\infty} \inf \frac{1}{N} \sum_{k=0}^{N} r(p_k).$$

In this section, we will discuss the class of games $\mathcal{G} = \{\mathcal{G}_j\} = \langle G, P\rangle$ where $\mathcal{G}_j = (G, j, P)$ is the game with start position j. Notice, the payoff function P actually depends on the reward function r for the game. This will be implicit throughout this section.

5.4.1 Notation

For a vector $u \in \mathcal{R}^k$, we say $u < v$ provided $u(i) \leq v(i)$ for each i, and $u(j) < v(j)$ for some j. The norm of u is $\|u\| = \max_i\{|u(i)|\}$. Given a function $r : V \longrightarrow \mathcal{R}$, where V is a finite set, and $W \subset V$, define $|W|$ to be the number of elements in the set W, $r(W) = \sum_{w\in W} r(w)$, and $mean(W) = \frac{r(W)}{|W|}$.

A map $T : \mathcal{R}^k \longrightarrow \mathcal{R}^m$ is said to be a contraction provided there is a $0 \leq \beta < 1$ such that $\|T(x) - T(y)\| \leq \beta\|x - y\|$ for all x, y. A contraction map has a unique fixed point (i.e., unique x such that $T(x) = x$). Furthermore, for any y $\lim_{n\to\infty} T^n(y) = x$ (by T^n, we mean the n fold composition of T). See [24, 37] for a proof of this.

The games are played on the game graph $G = (V, E)$ where $V = \{1, \ldots, 2n\}$. We remind the reader that $I = \{1, \ldots, n\}$, $V_1 = \{2i | i \in I\}$, and $V_2 = \{2i - 1 | i \in I\}$. The edges out of position j are $(j, e_0(j))$ and $(j, e_1(j))$; the set $N(j) = \{e_0(j), e_1(j)\}$. We will usually write τ for a strategy of Player 1 and σ for a strategy of Player 2.

5.4.2 1-Player Payoff Games

A 1-Player payoff game is $\mathcal{G}_\tau = (G_\tau, P)$ where $G_\tau = (V, E_2 \cup \tau)$. The strategy for Player 1 is fixed to τ and the game is played by Player 2. The reward function is $r : V \longrightarrow \mathcal{R}$. The 1-Player payoff games are deterministic analogues of Markov

decision problems [20, 24, 37]. We study the N-step game, and the infinite discounted and mean payoff games. We show that 1-Player discounted and mean payoff games have a value and optimal positional strategies. The value and optimal strategies in 1-Player games can be computed in polynomial time.

N-Step

The payoff is

$$P_N(p) = \sum_{k=0}^{N-1} \beta^k r(p_k) + \beta^N f(p_N)$$

where $f : V \longrightarrow \mathcal{R}$ is the terminal reward function.

Definition 5.4.1 *Define the map $T_\tau : \mathcal{R}^{2n} \longrightarrow \mathcal{R}^{2n}$ where for $i \in I$,*

$$T_\tau(u)(2i) = r(2i) + \beta u(\tau(2i)) \tag{5.2}$$

$$T_\tau(u)(2i-1) = r(2i-1) + \beta \min_{a \in N(2i-1)} \{u(a)\}.$$

Equation 5.2 is the optimality equation in dynamic programming.

Lemma 5.4.1 *The value of the N-step 1-Player game $\mathcal{G}_\tau$ is $u_N = T_\tau^N(f)$.*

Discounted Payoff

The payoff is

$$P_\beta(p) = \sum_{k=0}^{\infty} \beta^k r(p_k)$$

where $\beta \in (0, 1)$.

Lemma 5.4.2 *The map $T_\tau : \mathcal{R}^{2n} \longrightarrow \mathcal{R}^{2n}$ in Definition 5.4.1 satisfies*

$$\|T_\tau(u) - T_\tau(v)\| \leq \beta \|u - v\|.$$

Therefore for $\beta \in (0, 1)$, T_τ is a contraction and has a unique fixed point.

Lemma 5.4.3 *The value of the game $\mathcal{G}_\tau$ is v, where $v = T_\tau(v)$. Player 2 has an optimal positional strategy σ where for $i \in I$,*

$$\sigma(2i-1) = \arg \min_{a \in N(2i-1)} \{v(a)\}.$$

Mean Payoff

The payoff in the mean payoff game is

$$\bar{P}(p) = \lim_{N\to\infty} \inf \sum_{k=0}^{N} r(p_k).$$

It is easy to see that the value of position i in game $\mathcal{G}_\tau$ is the smallest mean value of a cycle that is reachable from i. Furthermore, Player 2 has an optimal positional strategy which drives the game from position i to this cycle.

5.4.3 A Finite N-Step Game

In a finite N-step Game $\mathcal{G} = (G, P_N)$, the game ends after N steps. The play of the game is $p = p_0 p_1 \dots p_N$. Player 2 pays to Player 1

$$P_N(p) = \sum_{k=0}^{N-1} \beta^k r(p_k) + \beta^N f(p_N)$$

where β is a discount factor, and $f : V \longrightarrow \mathcal{R}$ is the terminal reward function. We want to determine the value v_N of the N-step game, and the optimal strategies of the two players.

Definition 5.4.2 *Define the map $T : \mathcal{R}^{2n} \longrightarrow \mathcal{R}^{2n}$ where for $i \in I$,*

$$T(u)(2i) = r(2i) + \beta \max_{a\in N(2i)} \{u(a)\} \tag{5.3}$$

$$T(u)(2i-1) = r(2i-1) + \beta \min_{a\in N(2i-1)} \{u(a)\}$$

Equation 5.3 is the basic optimality equation for the payoff games. Using dynamic programming, we can determine the value of the game, and the strategies of the players. The strategy for Player 1 is $\{\tau_k\}$ where $\tau_k : V_1 \longrightarrow V_2$. At step k, when the game is in position p_k and it is Player 1's turn to move, he moves to $\tau_k(p_k)$. Strategy for Player 2 is similarly defined.

Lemma 5.4.4 *The value of the N-step game with terminal reward f is $v_N = T^N(f)$. The optimal strategy for Player 1 is $\{\tau_k\}$ where*

$$\tau_{N-k}(2i) = \arg \max_{a\in N(2i)} \{v_{k-1}(a)\}.$$

The optimal strategy for Player 2 is $\{\sigma_k\}$ *where*

$$\sigma_{N-k}(2i-1) = \arg \min_{a \in N(2i-1)} \{v_{k-1}(a)\}.$$

Proof: By induction on N. ∎

Example 5.4.1 *Let's consider the payoff game in Figure 5.5 where the game stops after* $N = 5$ *steps. The discount factor* $\beta = 1$, *and the terminal reward is* 0 *at each position. The strategy for Player 1 is* $\{\tau_k\}$, *and for Player 2,* $\{\sigma_k\}$, *where* $k = 0, \ldots, 4$. *We leave it to the reader to show that for each* k $\tau_k(A) = F$, $\tau_k(C) = D$, *and* $\tau_k(E) = F$; *and* $\sigma_k(B) = C$, $\sigma_k(D) = C$, *and* $\sigma_k(F) = A$. *When the game starts from position A, Player 2 wins the game.*

Next consider the game for a large number of steps, say $N = 1000$. *We leave it to the reader to show that when the game starts from position A, Player 1 wins the game.*

5.4.4 Discounted Payoff Game

In this section, we study the discounted payoff game (DPG) $\mathcal{D} = (G, P_\beta)$, where discount factor $\beta \in (0, 1)$. The play is $p = p_0 p_1 p_2 \ldots$, and the payoff is

$$P_\beta(p) = \sum_{k=0}^{\infty} \beta^k r(p_k).$$

We prove Shapley's result that the game has a value, and both players have optimal positional strategies. We also discuss Howard's policy iteration algorithm [20] and its extension to stochastic games [19].

Value of the Game and Successive Approximation

Lemma 5.4.5 and Theorem 5.4.1 are due to Shapley [38].

Lemma 5.4.5 *The map* T *in Definition 5.4.2 satisfies*

$$\|T(u) - T(v)\| \leq \beta \|u - v\|.$$

Proof: For $j \in V_1$,

$$\begin{aligned} |T(u)(j) - T(v)(j)| &= \beta |\max_{a \in N(j)} \{u(a)\} - \max_{a \in N(j)} \{v(a)\}| \\ &\leq \beta \max_{a \in N(j)} |u(a) - v(a)| \\ &\leq \beta \|u - v\| \end{aligned}$$

Similarly for $k \in V_2$, $|T(u)(k) - T(v)(k)| \leq \beta\|u - v\|$. Therefore $\|T(u) - T(v)\| \leq \beta\|u - v\|$. ∎

By Lemma 5.4.5, the map T in Definition 5.4.2 is a contraction, and hence has a unique fixed point v.

Theorem 5.4.1 *The value of the game the DPG $\mathcal{D}$ is v where $v = T(v)$.*

Proof: Define the strategy τ for Player 1 as:

$$\tau(2i) = \arg \max_{a \in N(2i)} \{v(a)\}.$$

Similarly, the strategy σ for Player 2 is:

$$\sigma(2i-1) = \arg \min_{a \in N(2i-1)} \{v(a)\}.$$

From Lemma 5.4.3, σ is the optimal response of Player 2 to τ, and τ is the optimal response of Player 1 to σ. Therefore, strategies τ and σ are optimal in (MD, MD). From Lemma 5.2.1, they are also optimal in (HD, HD). The payoff when Player 1 plays with τ, and Player 2 plays with σ is v. ∎

How do we compute the value of the discounted payoff game? This value can be approximated by computing the value of the N-step game for large N. This is known as the method of successive approximation [38, 20, 37, 24].

Lemma 5.4.6 *Suppose v is the value of the discounted payoff game. Then $\lim_{N \to \infty} v_N = v$ where v_N is the value of the N-step game.*

Proof: Since $v_N = T^N(f)$ and $\lim_{N \to \infty} T^N(f) = v$, where v is the unique fixed point of T. ∎

Theorem 5.4.2 *Consider the discounted payoff game $\mathcal{D} = (G, P_\beta)$ with $G = (V, E)$ and $\beta = \frac{k}{l}$. Then using the successive approximation technique, the value of the DPG $\mathcal{D}$ can be computed in $O(\frac{n^2 log(l) + log M}{log(\frac{l}{k})})$ steps where $|V| = 2n$ and $M = \max_{v \in V} |r(v)|$.*

Proof: First we will show that the value of position i, $v(i)$, is a rational number $\frac{n}{d}$ where n and d are integers. Then we will get a bound on the size of d.

The discounted game has optimal positional strategies for the two players. When the two players play these positional strategies, the play from position i is $p = p_0 p_1 \dots p_g (p_{g+1} \dots p_{g+h})^\omega$ where $g + h \leq 2n$. Therefore the value

$$v(i) = \sum_{m=0}^{g} \beta^m r(p_m) + \frac{\beta^g}{1 - \beta^h} (\sum_{j=1}^{h} \beta^j r(p_{g+j})).$$

The value of position i can be written as a rational number $v(i) = \frac{n'}{d'}$ where $d' = l^{g+h}(l^h - k^h)$. But since g and h are not known a priori, we define

$$d = l^{2n} \prod_{j=1}^{2n} (l^j - k^j).$$

For each i, we can write $v(i)$ as a rational number with denominator d.

Since

$$\|T^N(0) - T^\infty(0)\| \leq \frac{\beta^N}{1 - \beta} M,$$

we choose N such that

$$\frac{1}{2d} > \frac{\beta^N}{1 - \beta} M.$$

This gives us N is $O(\frac{n^2 log(l) + log M}{log(\frac{l}{k})})$. The value $v(i)$ in the DPG $\mathcal{D}$ is obtained by computing $v_N(i)$ in the N-step game and rounding it to to the nearest rational with denominator d. ∎

From Theorem 5.4.2, it follows that for a fixed β, the discounted payoff games can be solved in polynomial time. Unfortunately, this method can be very slow in converging when $\beta \approx 1$ (see Example 5.4.1). To overcome the problem with slow convergence, we will study the policy iteration algorithm.

Strategy Improvement using Policy Iteration

In this section, we will show how to use a variant of the policy iteration algorithm [20, 37, 24] to determine the value of the discounted payoff game $\mathcal{D} = (G, P_\beta)$.

Definition 5.4.3 *Let us fix positional strategy τ for Player 1, and positional strategy σ for Player 2 in the DPG $\mathcal{D}$. Define the map $T_{\tau\sigma} : \mathcal{R}^{2n} \longrightarrow \mathcal{R}^{2n}$ by*

$$T_{\tau\sigma}(u)(2i) = r(2i) + \beta u(\tau(2i)) \tag{5.4}$$

$$T_{\tau\sigma}(u)(2i-1) = r(2i-1) + \beta u(\sigma(2i-1)).$$

Lemma 5.4.7 *The map $T_{\tau\sigma}$ in Equation 5.4 has the following properties:*

1. *For $u < v$, $T_{\tau\sigma}(u) < T_{\tau\sigma}(v)$.*
2. *$\|T_{\tau\sigma}(u) - T_{\tau\sigma}(v)\| \leq \beta\|u - v\|$.*

Hence, $v = T_{\tau\sigma}(v)$ is the unique fixed point. The vector v is the payoff when Player 1 plays strategy τ, and Player 2 plays strategy σ in game $\mathcal{D}$. The value v can be computed in polynomial time by solving the system of linear equations in Equation 5.4. We are now ready to describe the strategy improvement algorithm of [19].

Strategy Improvement Algorithm:

1. *$i = 0$; Player 1 picks an initial positional strategy τ_0.*
2. *repeat*
3. *Player 2 responds with positional strategy $\sigma_i \in r_2(\tau_i)$.*
4. *Compute the fixed point $v_i = T_{\tau_i\sigma_i}(v_i)$.*
5. *For $j \in I$, define $\tau_{i+1}(2j) = \tau_i(2j)$ when $\tau_i(2j) \in \arg\max_{a \in N(2j)}\{v_i(a)\}$, else define $\tau_{i+1}(2j) = \arg\max_{a \in N(2j)}\{v_i(a)\}$.*
6. *$i := i + 1$*

7. *until*$(\tau_i = \tau_{i-1})$

8. $\tau^* = \tau_{i-1}$; $\sigma^* = \sigma_{i-1}$.

At each step τ_i and σ_i are positional strategies. Each iteration of the algorithm takes only polynomial time. We next show that the algorithm incrementally improves the strategy for Player 1. That is, strategy τ_{i+1} is a better strategy than τ_i.

Theorem 5.4.3 *Suppose* $v_i = T_{\tau_i\sigma_i}(v_i)$ *and* $v_{i+1} = T_{\tau_{i+1}\sigma_{i+1}}(v_{i+1})$. *Then* $v_i < v_{i+1}$.

Proof: Since

$$\sigma_i(2j-1) = \arg \min_{b \in N(2j-1)} \{v_i(b)\},$$

and

$$\tau_{i+1}(2j) = \arg \max_{a \in N(2j)} \{v_i(a)\},$$

we get $v_i < T_{\tau_{i+1}\sigma_{i+1}}(v_i)$. From Lemma 5.4.7, $v_i < T_{\tau_{i+1}\sigma_{i+1}}(v_i) < T^2_{\tau_{i+1}\sigma_{i+1}}(v_i) < \cdots < T^n_{\tau_{i+1}\sigma_{i+1}}(v_i)$. But $\lim_{n\to\infty} T^n_{\tau_{i+1}\sigma_{i+1}}(v_i) = v_{i+1}$. Hence $v_i < v_{i+1}$. ∎

The strategy for Player 1 is improving at each step. In a game with $|V_1| = n$, Player 1 has atmost 2^n strategies. Therefore the algorithm terminates in at most 2^n steps with optimal strategies τ^* and σ^*.

Example 5.4.2 *Let us look at the payoff game in Figure 5.5 with discounted payoff and discount factor* $\beta = 0.999$. *We only need to look at what Player 1 does at position A, and what Player 2 does at position D since the moves at the other positions are fixed. We will solve the game using the strategy improvement algorithm. The initial strategy for Player 1 is* τ_0 *where* $\tau_0(A) = F$. *Then* $\sigma_0 \in r_2(\tau_0)$ *where* $\sigma_0(D) = E$. *The value is* v_0 *where* $v_0(A) = -499.25$, $v_0(B) = -398.35$, $v_0(C) = -298.65$, $v_0(D) = -300.95$, $v_0(E) = -300.25$ *and* $v_0(F) = -500.75$. *Since* $v_0(B) > v_0(F)$, $\tau_1(A) = B$. *Then* $\sigma_1(D) = C$, $v_1(A) = 400.85$, $v_1(B) = 400.25$, $v_1(C) = 500.75$, $v_1(D) = 499.25$, $v_1(E) = 598.05$, *and* $v_1(F) = 398.45$. *At the next iteration* $\tau_2 = \tau_1$ *and the algorithm stops. The reader can compare this with the successive approximation algorithm from Example 5.4.1.*

5.4.5 Mean Payoff Games

We next study the mean payoff game (MPG) $\mathcal{M} = (G, \bar{P})$ in which the play is $p = p_0 p_1 p_2 \ldots$ and the payoff is

$$\bar{P}(p) = \lim_{N \to \infty} \inf \frac{1}{N} \sum_{k=0}^{N} r(p_k).$$

Ehrenfeucht and Mycielski [10] showed that mean payoff games have a value and both players have optimal positional strategies. To do this, they needed to work with both finite and infinite games. They define a finite game in which the play stops as soon as a cycle is formed. Player 2 then pays to Player 1 the mean value of the cycle. All finite games have a value [25]. Ehrenfeucht and Mycielski show that this is also the value of the infinite game. To show that the game has optimal positional strategies, they needed to use the infinite game to show some results about the finite game. The proof is somewhat involved. Mycielski [28] mentions that no direct proof is known. In contrast, Shapley's proof for discounted payoff games relies on a contraction map, and is simpler. We give a new proof that mean payoff games have optimal positional strategies. In the proof, we make precise the relationship between discounted and mean payoff games.

Theorem 5.4.4 *Consider a game graph $G = (V, E)$ with the reward function $r : V \longrightarrow \mathcal{Z}$. Suppose v_β is the value of the DPG $\mathcal{D} = (G, P_\beta)$. Then*

1. *There is a $0 < \beta^* < 1$ and positional strategies τ^* and σ^* such that for all β where $\beta^* \leq \beta < 1$, τ^* and σ^* are optimal in the DPG $\mathcal{D} = (G, P_\beta)$.*

2. *Strategies τ^* and σ^* are optimal for the MPG $\mathcal{M} = (G, \bar{P})$, and the value of the MPG is $\bar{v} = \lim_{\beta \to 1}(1 - \beta)v_\beta$.*

Proof: 1) Fix a start position. Let τ and σ be positional strategies and $p = fc^\omega = p_0 p_1 \ldots p_g (p_{g+1} \ldots p_{g+h})^\omega$ the resulting play when Player 1 plays τ and Player 2 plays σ. Then $g \leq |V| - 1$, $|h| \leq |V|$,

$$P_\beta(p) = \sum_{m=0}^{g} \beta^m r(p_m) + \frac{\beta^g}{1 - \beta^h} (\sum_{j=1}^{h} \beta^j r(p_{g+j})),$$

and

$$(1-\beta)P_\beta(p) = (1-\beta)\sum_{m=0}^{g}\beta^m r(p_m) + \frac{\beta^g}{\sum_{l=1}^{h}\beta^{l-1}}(\sum_{j=1}^{h}\beta^j r(p_{g+j})).$$

Thus $\lim_{\beta\to 1}(1-\beta)P_\beta(p) = mean(c)$. Let τ' and σ' be some other positional strategies with play $p' = f'(c')^\omega$. Consider

$$poly(\beta) = ((1-\beta)P_\beta(p) - (1-\beta)P_\beta(p'))(\sum_{l=1}^{h}\beta^{l-1})(\sum_{l'=1}^{h'}\beta^{l'-1}),$$

where $h = |c|$ and $h' = |c'|$. Then $poly(\beta)$ has at most $2|V|$ zeroes since it is a polynomial of degree at most $2|V|$. Thus the strategy pair (τ,σ) and (τ',σ') have the same value at most $2|V|$ times. When (τ,σ) is optimal at discount factor β_1, and (τ',σ') is optimal at β_2, it must be the case that two strategy pairs have the same value at some α for $\beta_1 < \alpha < \beta_2$, but not the same value at β_1. Since there are only $2^{|V|}$ different strategy pairs, this can only happen a finite number of times. Therefore there is a β^* where $0 < \beta^* < 1$, and positional strategies τ^* and σ^* such that for all β, where $\beta^* \leq \beta < 1$, τ^* and σ^* are optimal in the DPG $\mathcal{D} = (G, P_\beta)$.

2) Let the resulting play from playing τ^* and σ^* be $p = fc^\omega$. We first show that τ^* and σ^* are optimal in *(MD,MD)* for the MPG $\mathcal{M} = (G, \bar{P})$. Suppose σ', not σ^*, is the optimal response of Player 2 to τ^*. Let $p' = f'(c')^\omega$ be the resulting play. Then $mean(c') < mean(c)$. Therefore $\lim_{\beta\to 1}(1-\beta)P_\beta(p') < \lim_{\beta\to 1}(1-\beta)P_\beta(p)$. But then for some $\beta^* < \beta' < 1$, $P_{\beta'}(p') < P_{\beta'}(p)$ and σ' is the optimal response for Player 2 in the DPG $\mathcal{D} = (G, P_{\beta'})$ — a contradiction. Similarly τ^* is the optimal response to σ^*. Thus the MPG has the value

$$\bar{v} = mean(c) = \lim_{\beta\to 1}(1-\beta)v_\beta$$

in *(MD,MD)*. From Lemma 5.2.1 and since *MD* is complete for 1-Player mean payoff games, $\bar{v}$ is also the value in *(HD,HD)*. ∎

As our intuition would suggest, the optimal strategies in the MPG are the same ones which would be used for the DPG for β close to 1. In [43], it is shown how the value of the mean payoff game can be obtained from solving the discounted payoff game. It is also possible to extend the successive approximation and the strategy improvement algorithms directly to mean payoff games.

5.4.6 Complexity

Let us consider the computational complexity of solving payoff games. For a MPG $\mathcal{M} = (G, p_0, \bar{P})$, consider the following question: does Player 1 win $\mathcal{M}$? The problem is in NP because the optimal strategy τ for Player 1 can be "guessed". Then one finds the value of the 1-Player game $\mathcal{M}_\tau$ — a polynomial time problem. The "complement" of the problem — does Player 1 not win the game — is equivalent to showing that Player 2 wins the game. The "complement" problem is also in NP since determining if Player 2 wins is in NP. Therefore the problem is in $NP \cap co-NP$. Notice, to show this, essential use is made of three facts: the game has a value, both players have optimal positional strategies (and therefore the optimal strategies are of polynomial size), and 1-Player games are solvable in polynomial time. By similar reasoning, it can be shown that determining whether Player 1 wins a discounted payoff game is in $NP \cap co - NP$.

At present, no polynomial time algorithm is known for solving payoff games. Our computational experience with the policy iteration algorithm suggests that it is an efficient algorithm for solving mean payoff games and discounted payoff games. At present, it is not known whether the policy iteration algorithm is a polynomial time algorithm.

The complexity question for the problem has also attracted attention in the computer science community [9, 26, 43]. Condon [9] studies a simplified version of stochastic games called *simple stochastic games.* She points out that this problem is in $NP \cap co - NP$. Zwick and Patterson [43] study mean payoff and discounted payoff games. They provide a polynomial reduction from payoff games to simple stochastic games, hence showing that these games are perhaps easier than payoff games. Ludwig [26] gives an interesting randomized algorithm of complexity $O(2^{\sqrt{|V|}})$ for solving simple stochastic games. The same algorithm extends to mean payoff games and discounted payoff games.

5.5 Chain Games and Mean Payoff Games

A chain game has n pairs — (B_i, G_i) — where the sets B_i and G_i are disjoint. Player 1 wins the play p provided $\inf(p)$ "touches" G_i but not B_j for any $j \geq i$.

In this section, we show how to translate a chain game into a mean payoff game, so that both games have the same winner. Using this reduction, it becomes possible to use the successive approximation algorithm and the policy iteration algorithm to solve chain games.

In Section 5.5.1, we will give the method to translate a chain game into a mean payoff game. We will prove that the same player wins both the chain game and the mean payoff game. We then discuss the consequences of this result.

5.5.1 Reducing a Chain Game to a Mean Payoff Game

The new mean payoff game will be played on the same game graph as the chain game. We assign positive rewards to positions in G_i, and negative rewards to positions in B_i. The idea is to assign much larger positive rewards to positions in G_i, than positions in B_j for $j < i$; and much larger negative rewards to positions in B_k for $k \geq i$. So Player 1 wins the mean payoff game iff the play visits a position in G_i, and perhaps some positions in B_j for $j < i$, but no position in B_k for $k \geq i$.

Example 5.5.1 *A Buchi Game has a set $F \subset V$ of final states. Define the reward function where $r(f) = 1$ for $f \in F$, and $r(k) = 0$ for $k \notin F$. Then Player 1 wins the Buchi Game iff the value of the mean payoff game is v where $v > 0$. It is easy to see that the positional strategy which wins one game also wins the other.*

Let us now discuss the general method for translating a chain game into a mean payoff game. The reward function is $r : V \longrightarrow \mathcal{Z}$ where positive rewards are assigned to position in G_k, and negative rewards are assigned to positions in B_k. For $g \in G_k$, the reward is

$$r(g) > |\sum_{j=0}^{k-1} r(B_j)|.$$

For $b \in B_k$, the reward is

$$r(b) < -|\sum_{j=1}^{k} r(G_j)|.$$

When the reward function satisfies the inequalities given above, both the chain game and mean payoff game have the same winner. We next define such a reward function.

Translating a chain game into a mean payoff game:
Given a chain game $\mathcal{G} = (G, P_\phi)$ with N positions and m pairs (B_i, G_i), define the mean payoff game $\mathcal{M}_\mathcal{G} = (G, \bar{P})$ where the reward function is defined as follows:
For $g \in G_k$,

$$r(g) = N^k,$$

and for $b \in B_k$,

$$r(b) = -N^{k+1}.$$

Theorem 5.5.1 *The same player wins the chain game $\mathcal{G}$ and the mean payoff game $\mathcal{M}_\mathcal{G}$. Furthermore, the same positional strategy can be used to win both games.*

Proof: First let us show that for a cycle C, $\phi[C] = 1$ iff $r(C) > 0$. Suppose $\phi[C] = 1$ and the position with the largest index in C is v. Then $v \in G_k$ and when $B_j \cap C \neq \emptyset$, $j < k$. Therefore

$$r(C) = \sum_{c \in C} r(c) = r(v) + \sum_{w \in C \setminus \{v\}} r(w).$$

But

$$\sum_{w \in C \setminus \{v\}} r(w) \geq -(N-1)N^{k-1} > -N^k.$$

Since $r(v) = N^k$, $r(C) > 0$. Similarly, when $r(C) > 0$, $\phi[C] = 1$.

Player 1 wins the chain game $\mathcal{G}$ using positional strategy τ iff for every cycle C in G_τ reachable from start position p_0, $\phi[C] = 1$. But $\phi[C] = 1$ iff $r(C) > 0$. And $r(C) > 0$ for every cycle C in G_τ reachable from p_0 iff Player 1 wins the mean payoff game $\mathcal{M}_\mathcal{G}$ with strategy τ. Therefore Player 1 wins the chain game $\mathcal{G}$ using positional strategy τ iff he wins the mean payoff game $\mathcal{M}_\mathcal{G}$ with the same strategy. ∎

5.5.2 Some Consequences

Theorem 5.5.1 has interesting consequences. It states that a chain game is a special instance of the mean payoff game. But any game on an ω-automaton may be translated to a chain game (Theorem 5.3.2). Therefore any game on an ω-automaton can be translated to a chain game and solved as a a mean payoff game. In particular, the policy iteration algorithm may be used to solve games on ω-automata. We also get a new algorithm for model checking the propositional μ-calculus using the policy iteration algorithm.

Many of the results we discussed in Section 5.3 about games on ω-automata become special cases of the results for mean payoff games. In particular Church's questions regarding solvability and synthesis are special cases of similar questions for mean payoff games. Also the fact that MD is complete for chain games follows from the result that MD is complete for mean payoff games of which chain games are a special instance.

Bibliography

[1] R. Abraham, J.E. Marsden, and T. Raitu, *Manifolds, Tensor Analysis, and Applications*, Springer-Verlag, 1988.

[2] R. Alur et. al., The Algorithmic Analysis of Hybrid Systems, *Theoretical Computer Science*, Feb. 1995.

[3] R.Alur, C.Courcoubetis, T.A. Henzinger and P.-H. Ho, Hybrid automata: an algorithmic approach to the specification and analysis of hybrid systems,*Hybrid Systems*, LNCS 736, Springer-Verlag 1993.

[4] R. Alur and D. Dill, Automata for modeling real-time systems, *Proc. 17th ICALP*, Lecture Notes in Computer Science 443, Springer-Verlag, 1990.

[5] J.P. Aubin and A. Cellina, *Differential Inclusions*, Springer-Verlag, 1984.

[6] J.P. Aubin, *Viability Theory*, Birkhauser, 1991.

[7] V. Borkar and P. Varaiya, ϵ-Approximation of Differential Inclusion using Rectangular Differential Inclusion, Notes.

[8] A. Church, Logic, arithmetic and automata, *Proc. International Congress of Mathematicians*, 1963.

[9] A. Condon, The complexity of stochastic games, *Information and computation*, 96:203-224, 1992.

[10] A. Ehrenfeucht and J. Mycielski, Positional strategies for mean payoff games, *International Journal of Game Theory*, 1979.

[11] E. A. Emerson, Automata, tableaux, and temporal logics, *Logics of Programs*, LNCS 193, Springer-Verlag, 1985.

[12] E. A. Emerson and C.S. Jutla, The complexity of tree automata and logics of programs, *Proc. of the 29th Ann. IEEE Symposium on Foundations of Computer Science*, 1988.

[13] E.A. Emerson, C.S. Jutla, and A.P. Sistla, On model-checking for fragments of μ-calculus, *Proc. of Fifth Conference on Computer Aided Verification*, LNCS 697, Springer-Verlag, 1993.

[14] J.Frankel, L.Alvarez, R.Horowitz, and P.Li. "Robust Platoon Manuevers for AVHS," UCB-PATH TECH NOTE 94-09, University of California.

[15] D.Godbole and J.Lygeros, Longitudinal Control of the Lead Car of a Platoon. *IEEE Transactions on Vehicular Technology*, 43(4):1125-35, November 1994.

[16] J. Guckenheimer and P. Holmes, *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*, Springer-Verlag, 1983.

[17] T. Henzinger, P. Kopke, A. Puri and P. Varaiya, What's Decidable About Hybrid Automata, *Proceedings of the 27th Ann. ACM Symposium on the Theory of Computing*, 1995.

[18] M. W. Hirsh and S. Smale *Differential Equations, Dynamical Systems, and Linear Algebra*, Academic Press, Inc., 1974.

[19] A.J. Hoffman and R.M. Karp, On Non-terminating Stochastic Games, *Management Science*, 12:359-370, 1966.

[20] R.A. Howard, *Dynamic Programming and Markov Processes*, M.I.T. Press, 1960.

[21] A.Hsu, F.Eskafi, S.Sachs, and P.Varaiya. Protocol Design for an Automated Highway System. *Discrete Event Dynamic Systems: Theory and Applications*, vol.2,(no.3-4):183-206, February 1993.

[22] D. Kozen, Results on propositional μ-calculus, *Theoretical Computer Science*, Dec. 1983.

[23] S.C. Krishnan, A. Puri, R.K. Brayton, and P.P. Varaiya, The Rabin index and chain automata, with applications to automata and games, *Proc. of the Seventh Conference on Computer Aided Verification*, LNCS 939, Springer-Verlag, 1995.

[24] P.R. Kumar and P. Varaiya, *Stochastic Systems: Estimation, Identification and Adaptive Control*, Prentice Hall, 1986.

[25] R.D. Luce and H. Raiffa, *Games and Decisions: Introduction and Critical Survey*, Dover Publications, 1957.

[26] W. Ludwig, A subexponential randomized algorithm for the simple stochastic game problem, *Information and Computation*, 117:151-155, 1995.

[27] R. Milner, *Communication and Concurrency*, Prentice Hall, 1989.

[28] J. Mycielski, Games with Perfect Information, *Handbook of Game Theory*, vol. 1, Elsevier Science, 1992.

[29] X.Nicollin, A.Olivero, J. Sifakis and S. Yovine, An Approach to the Description and Analysis of Hybrid Systems, *Hybrid Systems*, LNCS 736, Springer-Verlag 1993.

[30] A. Puri and P. Varaiya, Decidability of Hybrid System with Rectangular Differential Inclusions, *CAV 94: Computer-Aided Verification*, Lecture Notes in Computer Science 818, pages 95-104. Springer-Verlag, 1994.

[31] A. Puri and P. Varaiya, Verification of Hybrid Systems using Abstractions, *Hybrid Systems II*, LNCS 999, Springer-Verlag, 1995.

[32] A. Puri, V. Borkar and P. Varaiya, ϵ-Approximation of Differential Inclusions, *Proceedings of the 34th IEEE Conference on Decision and Control*, 1995.

[33] A. Puri and P. Varaiya, Driving Safely in Smart Cars, California PATH Research Report UCB-ITS-PRR-95-24, July 1995.

[34] A. Puri and P. Varaiya, Decidable Hybrid Systems, To appear in *Computer and Mathematical Modeling.*

[35] M.O. Rabin, *Automata on Infinite Objects and Church's Problem*, volume 13 of *Regional Conf. Series in Mathematics*, 1972.

[36] T.E.S. Raghavan and J.A. Filar, Algorithms for Stochastic Games - A Survey, *ZOR - Methods and Models of Operations Research*, 35:437-472, 1991.

[37] S.M. Ross, *Introduction to Stochastic Dynamic Programming*, Academic Press, 1983.

[38] L.S. Shapley, Stochastic Games, *Proceedings National Academy of Sciences*, vol. 39, 1957.

[39] C. Sparrow, *The Lorenz Equations*, Springer-Verlag, 1982.

[40] W. Thomas, Automata on Infinite Objects, *Formal Models and Semantics*, volume B of Handbook of Theoretical Computer Science, Elsevier Science, 1990.

[41] P.Varaiya. Smart Cars on Smart Roads: Problems of Control. *IEEE Transactions on Automatic Control*, 38(2):195-207, February 1993.

[42] P. P. Varaiya, On the Trajectories of a Differential System, in A.V. Balakrishnan and L.W. Neustadt, editor, *Mathematical Theory of Control*, Academic Press, 1967.

[43] U. Zwick and M. Patterson, The complexity of mean payoff games on graphs, *COCOON '95*, LNCS, Springer-Verlag.