From Text to Life:
Optimized Pacemaker Therapy Through Formal Methods and Safe Reinforcement Learning

by

John W. Komp

B.S., Michigan Technological University, 1983

M.S., University of Minnesota, 2011

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

2024

Committee Members:

Ashutosh Trivedi, Chair

Majid Zamani

Fabio Somenzi

Michael Rosenberg

Zhihao Jiang

Komp, John W. (Ph.D., Computer Science)

From Text to Life:

Optimized Pacemaker Therapy Through Formal Methods and Safe Reinforcement Learning

Thesis directed by Prof. Ashutosh Trivedi

Since their invention in 1958, **implantable pacemakers** have revolutionized the treatment of cardiac diseases, significantly enhancing patient quality of life. With each subsequent generation, advances in miniaturization of electronics, batteries, and sensors have facilitated smaller, more capable pacemakers. These innovations have brought faster arrhythmia detection and introduced novel therapies, but they also come with challenges. Tailoring these complex therapies to individual patients often requires the expertise of highly skilled cardiac clinicians, leaving many patients with suboptimal care based on generalized models. Moreover, increasing device complexity extends development and verification times, delaying the availability of new therapies. Despite these challenges, patient safety remains paramount.

Recent advances in **reinforcement learning** (RL) offer promising solutions to address these challenges. RL has demonstrated the ability to discover innovative solutions to complex problems at scale. Applied to pacemaker design, RL offer a transformative paradigm for managing complexity, reducing development time, and enabling personalized therapies, all while upholding the highest safety standards. Through RL's scalable exploration capabilities, new therapies can be discovered, and existing ones can be tailored to meet individual patient needs. However, the safe application of RL hinges on the precise construction of the reward mechanisms—often expressed as finite-state machines known as **reward machines**—that guide the behavior of RL agents. Errors in defining these learning objectives can result in flawed training and incorrect system operation.

This dissertation addresses the critical problem of rigorously and unambiguously expressing the requirements for cardiac pacemakers in a form that can be effectively processed by RL algorithms. Formal logic provides a robust framework for specifying system behaviors, and automated

methods for translating these requirements into reward machines can significantly enhance the reliability of RL-based system design. A key contribution of this dissertation is the formalization of cardiac pacemaker requirements using a real-time formal logic called **Duration Calculus** (DC). We further explore techniques to translate these formal specifications into reward machines, ensuring that RL algorithms trained to maximize these rewards produce strategies that satisfy the original requirements.

While formal logic offers a precise method for defining RL learning objectives, the manual process of capturing requirements in this format is often labor-intensive and error-prone. To address this, the dissertation investigates the potential of AI technologies—such as **recurrent neural networks** and **large language models**—to extract formal logic and reward machines from expert demonstrations provided by skilled cardiac clinicians.

RL's exploratory nature can lead it to make incorrect actions leading to grave consequences. To enhance safety, we also develop approaches to restrict the choices available to RL agents using **safety shields** derived from formal requirements. These shields ensure that the strategies generated by RL algorithms comply with critical safety and performance criteria.

By integrating formal logic, reinforcement learning, and cutting-edge AI techniques, this dissertation establishes a robust foundation for the next generation of intelligent cardiac pacemaker design, while ensuring both safety and personalization.

Dedication

To Karen with love, for only she knows how far a man will go to avoid washing dishes.

Acknowledgements

I'm deeply indebted to Ashutosh Trivedi for his time and patience with me during this journey. Teaching me to think and talk like a researcher and not an engineer must have been trying. I only hope having a long time industry refugee as a student was an enjoyable experience. I must thank Mats Heimdahl at the University of Minnesota for encouraging me to start this journey. And John Carlis, also at the University of Minnesota, for his continued interest in my research and his corny jokes. We lost him way too soon. Toby Markowitz, Medtronic Bakken Fellow, for his many, tireless hours teaching me pacing therapy. And finally, but certainly not last, my research partners: Kalyani Dole, Ashutosh Gupta, Shankaranarayanan Krishna, Abhay Rajput, and Namrita Varshney from the Indian Institute of Technology Bombay; Maria Pacheco and Dananjay Srinivas from the University of Colorado, Boulder.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

Since their inception in 1959, implantable pacemakers have transformed cardiac disease treatment, enhancing patient quality of life through successive technological advancements in miniaturization and cardiac therapy. However, the increasing complexity of these devices presents challenges in development, verification, and personalized care, often relying on generalized patient population models that can lead to suboptimal outcomes. Reinforcement learning (RL) is emerging as a promising solution to these challenges by enabling innovative, scalable approaches to personalized therapy, reduced development timelines, and complexity management. Central to the safe and effective application of RL in pacemaker design is the precise construction of reward mechanisms, such as reward machines, which guide RL agents' behavior. *This dissertation focuses on addressing the critical issue of rigorously formalizing cardiac pacemaker requirements to ensure their effective integration with RL algorithms, paving the way for safer, more personalized cardiac therapies.*

1.1 Optimized Pacemaker Therapy Through Safe Reinforcement Learning

Implantable medical devices have revolutionized patient health, freeing patients from bulky external equipment that limited their daily activities, significantly improving their quality of life. The first such device, a cardiac pacemaker, was implanted in 1959 [56]. That simple device was the genesis of the implantable medical device industry now spanning a vast array of medical disciplines including cochlear ear implants, neurological stimulation for chronic pain reduction, brain stimulation to treat Parkinson's disease, and drug pumps for regulation of blood sugars. These

device continue to evolve smaller and more sophisticated sensors that are providing new sources of real-time data about the patient's disease state underutilized by today's therapies. Through in-situ, real-time analysis of data, current therapies can be continually optimized and new therapies devised providing patients a fuller, symptom free life.

1.1.1 Implantable Cardiac Pacemakers

The first pacemaker provided very basic support, sustaining patients suffering from low heart rates. Electronics technology was primitive, limiting what therapies could be provided while available batteries suitable for implant were large with limited charge capacity requiring frequent replacement. As a result, the devices were large and bulky; a cosmetic issue for some patients. Yet the devices were incredibly impactful, allowing patients freedom to live normal lives outside of a clinic. Technology has rapidly advanced since then. Modern pacemakers are a fraction of the original size while having 32 bit low power CPUs which allow for more complex therapies and battery lives extending the time between replacements to over twenty years for many patients.

While pacemaker technical capabilities have increased substantially from the original device, pacemaker therapies remain based on treating the average patient. Additionally, a patient's disease state changes as they age. Current pacemakers do not have the ability to detect and update their pacing parameters leaving the patient with degraded cardiac support until their next clinical visit, potentially six months to a year away.

With the accelerating advancements in electronics, sensors, data analysis, and artificial intelligence, patients and clinicians are driving a demand for even more complex devices that can provide therapy unique to the individual patient's disease state. Meeting this need requires shorter product development cycles and complex therapy algorithms while meeting significantly more stringent regulatory requirements for proving safety and efficacy.

1.1.2 Reinforcement Learning for Pacemaker Design

Reinforcement Learning (RL) [138] is a sampling-based optimization method where a learning agent discovers optimal solutions driven by scalar reward signals. It is well suited to the task of getting novel new therapies to market faster, optimized to the individual patient. RL has a history of finding creative solutions for many complex tasks such as playing Go [128], video games [109, 158], integrated circuit routing [106], and identifying cancer in X-rays [87]. Self driving automobiles [79] and autonomous drones [147, 159] have highlighted RL's innate ability to recognize patterns not obvious to humans in real-time data from multiple sources that result in novel control solutions for handling unexpected events.

Pacemakers have multiple real-time sensors that track a patient's coronary output and physical activity available for exploitation by RL's novel capabilities. Continuous analysis of this data enables new therapy options and optimization of existing therapies. As an optimization example, Mobitz II heart block may reduce ventricular beats by one-third leaving the patient feeling tired and prone to fainting. Current pacemakers provide the missing beat but at a slower rate than the patient's intrinsic rhythm, helpful but not optimal. With RL, the patient's actual rhythm can be identified and matched continuously, providing optimal cardiac output. As the patient's disease state changes, RL's learning ability allows for recognition of changing patient conditions, correctable through continual therapy modifications, ensuring appropriate treatment between clinical visits.

The success of RL relies on high-quality feedback in the form of scalar rewards. These rewards are often generated by an observer, commonly expressed in the form of finite state machines known as **reward machines** [72], which embodies knowledge of desired learning objective. Reward machines are typically designed manually and are prone to human error [47, 129]. Incorrect rewards can lead RL agents to learn unintended behaviors [35] and potentially incorrect medical treatment [115]. This dissertation addresses the critical problem of rigorously and unambiguously expressing the requirements for cardiac pacemakers in the form of reward machines that can be effectively processed by RL algorithms.

1.2 Thesis Statement

There is increasing demand for enhancing the capabilities of pacemakers, reducing development time, and providing adaptive therapy, all while maintaining the high patient safety standards required by the FDA. Recent progress in RL has demonstrated its potential for creative design exploration and adaptive control design at scale. However, the safe application of RL relies on the correct design of reward mechanisms that capture both safety requirements and non-functional specifications. Traditionally, formal methods offer a principled approach to designing safety-critical systems, emphasizing the importance of formal requirement specifications at every stage of the design process. This thesis develops the foundations for principled applications of RL in the design of implantable cardiac pacemakers, grounded in formal methods.

Thesis Statement

Reinforcement learning, grounded in formal methods, can be safely and effectively applied to the design of implantable cardiac pacemakers.

1.3 Contributions

The pacemaker and human heart form a complex, closed-loop system that is time-critical and requires real-time logic to express the pacemaker's requirements. Duration Calculus [30] (DC) is a highly expressive and succinct logic capable of capturing specifications involving dense-time durations. A key contribution of this thesis is the DC-based encoding of pacemaker requirements, as detailed in the Boston Scientific pacemaker specification document [86]. Subsequently, we focus on developing an automated approach to faithfully translate these requirements into reward machines and shields for RL agents, laying the foundations for a safe and effective RL-based approach to pacemaker design.

These key elements for enabling pacemaker therapy optimization are addressed in this thesis: translation of natural language specifications to formal specifications, formal specification of pacemakers, correct rewards machine synthesis, and safe, in-situ RL training.

1.3.1 Formal Specifications From Natural Language Explanations

While formal specifications have been shown to reduce design errors by eliminating the ambiguity of natural language while enabling rigorous testing methodologies [156, 113], most specifications are written in natural language. The manual translation process to a formal specification is difficult due to a lack of formal language proficiency among those doing the translation [62]. Automating the translation process provides a natural way to formalize natural language specifications to access the testing benefits of formal languages. This dissertation automates translation by combining the inherent language translation abilities of large language models, to extract candidate formal logic formula, with the logic synthesis of discounted semantics [7] to repair LLM mistakes.

1.3.2 Formal Pacemaker Specifications

Temporal formal logics provide a succinct language for expressing the evolution of systems over time. The language syntax of formal logic results in concise, grammatically well-formed formulas that can be converted directly to equivalent finite state machines that can be validated through well established formal testing methods such as deductive theorem proofs. Duration Calculus (DC) is a highly expressive logic that adds several unique temporal operators allowing capture of complex timing relationships. The expressiveness of DC's temporal operators, however, prevent translation to finite state machines. Through restriction of DC's temporal operators, subsets of DC regain translation to finite state machines. The first formal pacemaker specification is presented using restricted DC.

1.3.3 Reward Design for Safe Reinforcement Learning

This thesis presents and validates two reward machine creation methods that overcome the difficulties correct scalar reward design. First, a rewards machine can be extracted directly from the formal specifications. To achieve this, a translation process from specification to state machine will be explored. Secondly, it will be shown that a rewards machine can also be created using AI systems specializing in sequential data, recurrent neural networks and large language models. In this novel implementation expert demonstrations of desired environmental control are used for training an AI based rewards machine. This approach enables a unique new design process where experts directly define how the final design should work by capturing the expert's functional knowledge as traces or steps through a process.

1.3.4 Correct Reward Machines For RL Pacemaker Training

RL is not infallible, however. The Go playing RL was trained using the strategies of the Go grand masters. When an opponent used naive and simple strategies not in the original training data, the RL could easily be beaten [153]. Rewards machines, created from expert demonstrations or design specifications, used to train an RL controller may be incorrect or incomplete leaving gaps in the training process where the trained RL controller is required to make inferences. In a safety critical device, this would be unacceptable as there's no guarantee the chosen action is a safe action. Likewise, to optimize a patient's therapy, RL needs the ability to explore alternate therapy settings in order to observe the effects to the patient. If unconstrained, actions taken by the RL could easily cause patient harm.

Safety shields guard against inappropriate actions by the RL. The shield reviews each RL action, replacing those which are unsafe with an appropriate alternative that maintains safe operation. With a shield in place, the RL is free to explore where it's actions can never violate safety properties and incorrectly trained RL remain safe while optimizing patient therapy [55]. This will be demonstrated in this dissertation with a real-world example of pacemaker therapy optimization.

1.4 Thesis Organization

Patient safety is of paramount importance that cannot be overstated. Chapter 2 reinforces the need for avoiding patient risk by providing a historical background on software safety failures, regulatory oversight, and safety research.

Chapter 3 introduces the basics of the cardiac electrical system, how disease can disrupt proper operation and corrective pacing therapy. This cardiac and pacemaker background knowledge enables the reader to better understand the case study results.

Chapter 4 present an overview of Markov Decision Processes, automata, Formal logic, and reinforcement learning. It closes with an overview of the most common design process to stress how the design process has a direct impact on patient safety through error introduction.

Beginning with Chapter 5, each subsequent chapter builds on the previous in a journey from describing a process or therapy in a natural language to ultimately demonstrating a unique real world patient pacemaker therapy optimized to the individual patient.

Chapter 5 explores solutions to the error prone process of capturing desired performance from subject matter experts and formal requirements into verifiably correct representations.

Chapter 6 dives further into formal specifications through the introduction of a formal language capable of describing complex real-time behavior. To demonstrate utility of this approach, several formal language pacemakers specifications will be created and validated.

Chapter 7 addresses the issue of creating a rewards machine for RL training with two methods. First, a formal specification is converted to a rewards machine automaton. The second is unique in showing a neural net trained with expert demonstrations can be used as a rewards machine.

Chapter 8 completes the journey from specification to safe therapy with shielded RL. This chapter ends with a unique real-world demonstration of optimizing a pacemaker therapy.

1.5 Acknowledgment of Research Collaboration

This thesis is based on the results of four research papers focused on applications of AI in the specification and design of pacemakers. Each paper is a collaboration between researchers with unique skills in formal languages, reinforcement learning, and pacemaker design. Their contributions were instrumental and integral to the success of the research presented.

- *Event-triggered and time-triggered duration calculus for model-free reinforcement learning* [43]. This paper introduced subclasses of DC, used to create a single chamber pacemaker formal specification. Using this specification, the translation method from to a finite state machine based rewards machine was demonstrated.

- *Correct-by-construction reinforcement learning of cardiac pacemakers from duration calculus requirements* [44]. This paper extends the work of the previous paper to dual chamber pacemakers showing DC is capable of specifying complex timing relationships.

- *Integrating explanations in learning LTL specifications from demonstrations* [63]. This paper demonstrates extraction of formal requirements from natural language descriptions by combining large language models with robust semantics.

- *Show, don't tell: Learning reward machines from demonstrations for reinforcement learning-based cardiac pacemaker synthesis* [81]. This paper demonstrates creation of a rewards machine embodied as a sequential neural network from expert demonstrations.

Chapter 2

A Brief History of Medical Device Safety

Before beginning, a brief history on why patient safety is paramount and how the industry, research, and regulation have evolved over the years to address patient harm. The results have been mixed when using the standard design process. Applying RL with guarding to prevent errors may have a higher impact in creation of safe and effective treatment.

The story of the Therac-25 is a classic case history showing how a temporal software defect can escape detection during testing and cause severe patient harm in the field. Occurring from 1985 to 1987 and extensively documented in 1995 [91], the system overdosed multiple patients with x-rays. The failure was ultimately found to be caused by failure of the system specification and verification testing to account for unintended use by experienced users. While the user interface was specified and tested for appropriate input, the real-time nature of user input was not recognized. When the user interface was integrated with the system, unrecognized emergent properties were created. User input was signified by the return of the on-screen cursor to a home location. The cursor location would be periodically checked while background tasks continue to configure the x-ray source. Due to the length of time needed to perform some of these background tasks, an experienced user could move the cursor, edit dosage parameters and return the cursor to the home screen without system recognition of the user changes, leaving the system in an unsafe condition while all safety checks continued to indicate no safety violations.

Standard validation and verification testing was performed on Therac-25 with no failures while the emergent properties escaped into use. As can be found in Leveson's [91] failure analy-

sis, multiple parts of the software quality process failed to identify the emergent behavior. Key contributors to the failure of the Therac-25 were a failure to recognize and specify the temporal relationship between concurrently running software functions, poor software architecture, and a naive approach to the system hazard analysis.

While Therac-25 is a well documented example of medical device failure where the device software operated incorrectly or otherwise contributed to unexpected operation, it is not a solitary event. As documented by Wallace [152], over the years 1983 to 1997, from 2003 through 2012 [52], and from 2014–2018 [123], software errors continue to escape into use. From 1983–1991 medical device recalls due to software was 6%. This increased to an average of 10% over the years 1992–1997[2]. FDA analysis of recall data spanning 2008–2012 showed medical device software recalls had continued to increase to an average of 15% [152]. While not definitively proven, the reasons behind the steady increase are thought to be due to the increase in the use of software in medical devices, an improved failure reporting system, and increased vigilance by the world-wide medical regulatory bodies.

The US Food and Drug Administration, the FDA, uses a medical device classification system from Class I to Class III to denote patient risk of an adverse outcome using the device[1] . Class I is the lowest classification while Class III covers all devices that may cause severe adverse outcomes including death should a failure occur including devices and equipment that provide life sustaining support. Sarkissian et al. [123] looked at all causes of recalls for Class III devices over the span of 2014–1018. Function (29%) and Behavior (22%) were the most prevalent symptoms of software failure. When software failures were categorized by fault class, logic (43%) and calculation (24%) were the primary contributors.

Medical product manufacturers demonstrate product quality, efficacy, safety, and conformance to the FDA via a Design History File (DHF) in compliance with the Quality System Record (QSR) as codified in the Code of Federal Regulations, Title 21, Food and Drugs (21 CFR) part

[1] Medical device classification in Europe [42] is slightly different. See Rules 9, 10, and 11. This paper will use the FDA classification system

820 [53]. Documentation of the development process is described in §820.30 where the FDA defines in general how to perform and document design verification, conformance to product specifications (§820.30(f)), and validation, conformance to user requirements (§820.30(g)). The QSR defines what documentation is required to show compliance with 21 CFR. It does not define how to perform the steps of the process nor is software as a design component specifically addressed[2] .

Recognizing the impact of software on the safe operation of medical devices, in 2002, the FDA [50] published a guidance on software development and testing. This document specifies general good design practices for software development as taught in a university level software system engineering course. It recognizes the importance of user needs based requirements and product requirements along with testing to verify meeting those requirements. It does not define or specify how those requirements are obtained or shown to be complete.

In 2006 the International Electrotechnical Committee (IEC) began to address software quality in medical devices publishing IEC-62304 to cover the development of medical software. Like the FDA guidance, this document specifies a standard development process based on good system engineering principles. Again, like the FDA guidance, it does not define specifics on how to show requirements are complete and accurate nor address prevention of errors in design implementation.

Despite these initiatives, concern for faulty medical device software continued to grow. From 2005 through 2009 there were 56,000 adverse events reported to the FDA for external infusion pumps leading to 87 device recalls [51]. 70 of those recalls were designated Class II with the remainder considered Class I, the most severe. FDA analysis of these failures showed deficient software design led to improper device operation or enabled incorrect operation by the user. This prompted the FDA to launch the "Infusion Pump Improvement Initiative" in 2010 [51]. The primary two elements established in the initiative focus on increased analysis of device safety during the design process through creation of safety assurance cases [78] and a joint effort between the FDA, device manufacturers and university researchers to design more robust devices, known as the Generic Infusion Pump project. The project's goal was to define a reference specification

[2] The European equivalent governing document is Council Directive 2017/745 [42]

for an external infusion pump whose safety properties could be validated through formal modeling. Project research was archived on the infusion pump initiative web site[3] . Contributions appear to have ended in 2015 with no further additions.

In 2007 the North American Software Certification Consortium (SCC) initiated the Pacemaker Formal Methods Challenge hosted by the McMaster University Software Quality Research Lab (SQRL)[4] . Boston Scientific contributed an older dual chamber model pacemaker specification to use as requirements for a generic device [86]. The aim of the challenge was to show how applications of formal methods could lead to higher integrity software. A seminar to review the results of the challenge to date was convened at Schloss Dagstuhl [104] In 2012. Covering the topics of formal method certification, model-based development, medical domain unique aspects, tooling and pragmatics, researchers involved in the challenge reported on their research and progress. The overarching results of the seminar showed that formal methods and tools were not yet at a maturity level sufficient for deployment in development and testing of medical devices in commercial practice.

Based on papers published, extending applications of formal methods from generic cyber physical systems specifically to medical device software was an active area of research during the ten year period of 2005–2015. The volume of papers published since then has diminished significantly.

[3] Generic Infusion Pump Project web site: `https://rtg.cis.upenn.edu/gip/`
[4] see project website at `http://sqrl.mcmaster.ca/pacemaker.htm`

Chapter 3

Foundations of Cardiac Pacing Therapy

The heart is the most important organ in the body. Without its continuous beats pushing oxygenated blood through the vascular system, life ceases. Cardiac disease diminishes the heart's ability function properly and can progress to the point where the heart fails. When disease affects the electrical pathways of the heart, cardiac implantable electronic devices (CIEDs), such as cardiac pacemakers and implantable cardioverter-defibrillators (ICDs) can be implanted to restore functionality and improve the patient's prognosis and QOL. These small, battery power devices monitor the electrical activity of the heart looking for signs of incorrect cardiac function, providing a corrective or supportive electrical signal to the myocardium when needed.

Since the invention of the first simple bradycardia pacemaker in the 1950's, CIED design has become increasing complex. Where the first devices were not programmable and simply provided a continuous beat in a single chamber, modern multi-chamber devices are highly programmable tailoring their functionality to the individual patient needs and physiology while providing a continually adaptive pacing rate as the patient goes about their daily activities where the natural heart rate will change based on physical exertion. Commensurate with this increase in complexity has been an increase in design requirements leading to specifications spanning hundreds of pages of natural language requirements covering all aspects of the provided therapy and safety protocols.

The goal of cardiac pacing therapy is to replace a biological component of the cardiac cycle that has failed, is operating intermittently, or to mitigate an incorrect conduction pathway that is causing irregular heartbeats. Pacemakers perform this by applying voltage to the heart at regular

time intervals that are coordinated with the cardiac cycle. Pacing at an incorrect point in that cycle can lead to an irregular heartbeat or patient death. The paths taken by pacing pulses through the heart are not electrical conductors where electrons move through a matrix as an electrical engineer would recognize but a complex biological mechanism gating ions between cells. To understand how a pacemaker works starts with an understanding of the cardiac cycle and the heart's electrical pathways. What follows is a brief overview of the heart conduction system and diseases that can disrupt normal operation. For a deeper understanding of pacemakers and the heart conduction system I refer readers to *Clinical Cardiac Pacing, Defibrillation and Resynchronization Therapy* [77] and *An Introduction to the Cardiac Action Potentials* [141].

3.1 The Cardiac Cycle

The myocardium is the muscular structure of the heart and is a chemically activated electrical circuit. It is composed of oblong and stacked brick-like myocyte cells, separated by lateral membrane channels through which potassium (K) and sodium (Na) can pass. The permeability of the membrane is regulated by calcium (Ca). Pacing charge is carried by K^+ and Na^+ ions, flowing from cell to cell along the membrane network, causing each cell to depolarize from its resting state of approximately 85-90mV when sufficient charge accumulates within the cell. Depolarization across a sheet of cells is primarily directional along the longitudinal axis through the intercalated discs located on the axial ends of the cardiomyocytes. The direction of depolarization is anisotropic conduction, transmitting three times faster axially than radially, moving downward from the atria through the ventricles and outward from the endocardium to the epicardium. The amount of charge needed to trigger a myocardium cell, called the action potential (AP), varies with the cell's location and function in the heart. Directionality and potential variances are important in maintaining a coordinated contraction of all the cells of each heart chamber.

The AP of a cell can be viewed as a transient voltage gradient passing through the myocardium described through five phases: 0 - upstroke (depolarization), 1 - early repolarization, 2 - plateau, 3 - final repolarization, and 4 - resting. These are depicted in Figure 3.1, showing the

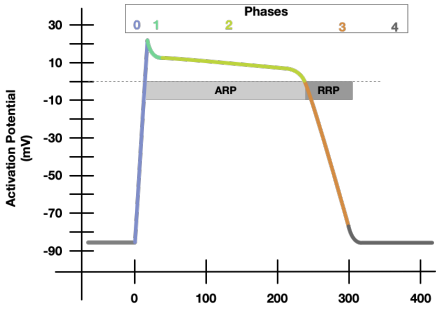


Figure 3.1: Ventricular Action Potential

morphology of the ventricular AP. The phases of the atrial AP are the same though their amplitudes and durations are shorter. Upstroke occurs when the cell membrane channel opens to allow an inrush of Na^+ ions to depolarize the negatively charged cell. The depolarization process will rapidly overshoot neutral charge becoming slightly positively charged, ending as the Na^+ membrane channel closes. Early repolarization is a rapid bleed off this overshoot as the cell moves back towards a negative charge. Before the positive overshoot has fully depleted, Ca regulation of the cell membrane leads to a charge plateau. Finally, the K^+ membrane channel opens and the cell rapidly completes its final repolarization returning to the initial negative charge. The cell remains in this resting state until the next depolarization cycle. The overall time from start of upstroke to return to rest is approximately 200-300ms [77, 122].

An important aspect of the action potential to pacing therapy is denoted in Figure 3.1 as absolute refractory period (ARP) and the relative refractory period (RRP). During the ARP the majority of the Na^+ channels are inactive thus preventing start of another AP. The RRP covers the majority of Phase 3 as more Na^+ channels recover and a new AP is possible, though requiring a higher Phase 0 amplitude correlated to the level inactive channels.

A region of specialized myocardium located at the top of the right atrium wall known as the sinoatrial node, or SA node, functions as the heart's natural pacemaker. It sets the heart rate by regularly triggering atrial depolarization. The cells in this region are a mix of atrial contractile and pacemaker cells, with the mix of pacemaker cells more dense towards the center of the region. Origin of the depolarization is not fully understood. The leading theories are DiFrancesco's Funny

Current, I_f, where this current grows to eventually push the SA node's AP over the threshold to trigger Phase 0 depolarization and Lakatta's theory of a calcium clock model where interactions between myocardial membranes and clusters of Ca receptors [141] cause a rise in the AP to exceed the threshold. Experimental results impairing either of these effects has shown impact to pacing function and both mechanisms may be involved.

The purpose of atrial contraction is to assure sufficient filling of the ventricles before the ventricles contract to push blood through the body. To coordinate the timing between the atrial and ventricular contractions, the pacing signal from the sinus node must be delayed before continuing to the ventricle to prevent a premature ventricular contraction (PVC) before ventricular filling is complete. Serving this purpose is the atrioventricular (AV) node, a small cardiac structure located between the right atrium and ventricle at the beginning of the ventricular conduction pathways or His bundle located in the intraventricular septum. The AV node provides a delay before starting the ventricular depolarization through a smaller resting potential and slower upstroke velocity (Na^+ inrush rate). This delay can be seen in the ECG as the time from the end of the P-wave to the start of the Q-wave. Like the SA node, the AV node has some pacemaker cells and can generate intrinsic ventricular beats in the absence of SA functionality.

When a group of cells are viewed as an aggregate, the action potential can be seen as a transient voltage gradient across the group. This gradient is what is displayed by surface electrocardiograms (ECG or EKG) measured across the torso, as depicted in Figure 3.2, and intracardiac electrograms (EGM) measured at a point directly on the myocardium. The cardiac cycle begins with contraction of the left and right atria. The P-wave seen in an ECG waveform is the rapid depolarization of the atria. Ventricular depolarization occurs next, designated as the QRS Complex in Figure 3.2. The final segment of a pacing pulse ECG is the T-wave. This represents the repolarization of the ventricles. Atrial repolarization occurs during ventricular depolarization and is not visible in ECG or EGM due to the much larger ventricular signal.

Allowing for only cell-to-cell propagation of the AP through the contractile myocytes would be too slow for coordinated contraction of a heart chamber. This is overcome with the cardiac

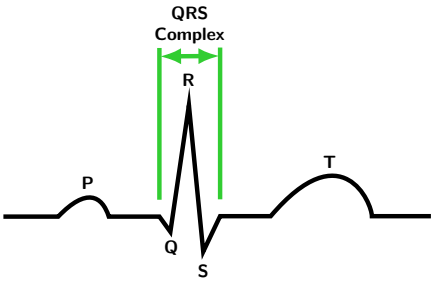


Figure 3.2: ECG of a single cardiac cycle

conduction pathways providing a high speed means to move the AP through the atria and ventricles ensuring the all the myocyte cells of each chamber beats together. These pathways are specialized cardiac cells that have lower thresholds to Phase 0 allowing them to quickly distribute the AP along their length. The Ventricular pathways are more organized and distinct with the interaction between the left and right bundle branches and the local myocytes being highly regulated through Purkinje fibers extending off the bundle branches into the endocardial tissue. Purkinje cells are highly resistive to receiving AP from the surrounding contractile myocytes to prevent retrograde conduction of the AP back into the conduction pathways.

3.2 Failures of the Heart

Coordinated propagation of AP through the myocardium is fundamental to heart health; disruption of this function leads to cardiac arrhythmias. Arrhythmias exhibit themselves as loss of intrinsic beats, lack of rate change due to activity or stress, slow rates, rapid rates, and extraneous beats. These conditions come from improper operation of the SA and AV nodes, impairment of AP conduction, and retrograde conduction.

Heart failure is a separate cardiac disease where the heart becomes enlarged and less efficient at pumping blood. The walls of the ventricles become thick and less contractile while coordination between the left and right ventricles decreases. Patients with heart failure typically exhibit an excess of sodium leading to water retention and additional health complications such as pulmonary edema and degraded quality of life.

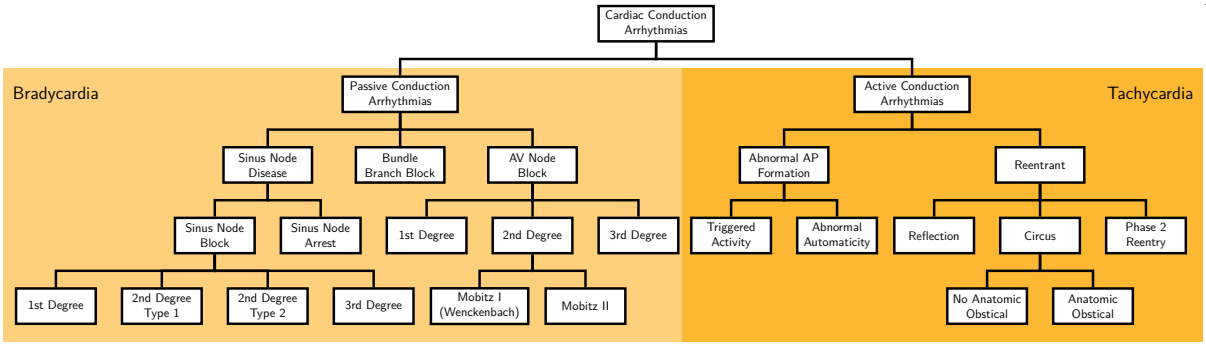


Figure 3.3: Categories of Cardiac Conduction System Arrhythmias

Coronary artery disease is a contributor to some forms of conduction arrhythmia as plaque builds in the coronary arteries. A myocardial infarction (heart attack) occurs when plaque build up causes a coronary artery to become occluded and prevent blood flow to a portion of the myocardium. If not treated rapidly, the impaired blood flow leads to cellular death (myocardial necrosis), leaving an infarcted region of tissue that is non-contractile and electrically non-conductive [49].

3.2.1 Conduction System Diseases

Cardiac arrhythmias have many causes including aging, congenital defects, viral infection, infarction, and coronary artery disease. Degradation or failure of the heart's conduction system can lead to arrhythmias where the heart can beat too slowly (bradycardia) or too quickly (tachycardia). Figure 3.3 categorizes the types of conduction failures that can lead to either or both of these conditions in a patient. The left half of the figure presents the ways that a pacing signal can fail to propagate leading to a slow heart rate or missing beats. The right half shows failures of the conduction system where the heart accelerates improperly due to overly excitable pacemaker functions or circular paths that allow an AP to continue propagating through the myocardium.

Understanding the nature of these conduction issues is important to pacing therapy for recognizing the type and cause of the arrhythmia and deciding the appropriate time to pace. Following Figure 3.3, this section is an overview of the bradycardia and tachycardia arrhythmias.

3.2.1.1 Bradycardia

By consensus, heart rates below 60 beats per minute are considered bradycardia [131]. A low heart rate is not necessarily unhealthy. Sleeping heart rates for healthy people can fall into the 50s or lower while physical fit people can strengthen their hearts, making them more efficient on each contraction thus requiring less beats to maintain needed blood flow resulting in normal bradycardia heart rates. For symptomatic patients, bradycardia rates do not supply sufficient blood to sustain activity and those patients may be lethargic, tire quickly or have episodes of fainting (syncope).

Bradycardia can be caused by failure of the SA node (sinus bradycardia) or the AV node (ventricular bradycardia). Reviewal of patient ECG will show missing P waves or QRS complexes depending on which type of failure is present. When there is complete failure of the natural pacemaker function or transmission of the resultant AP, a patient may exhibit an *escape rhythm* where other myocardial tissue containing pacemaker function cells generate a slower rate beat [38].

Sinus Node Disease. Sinus Bradycardia is the result of Sinus Node Disease (SND), a failure of the pacemaker function to properly maintain a constant beat at a rate necessary to sustain normal patient activity [59, 66]. SND may occur at any age and prevalence increases sharply with age. In the general population of 45 and older, the prevalence is 0.8 per 1000 (0.08%), increasing to 1 in 600 (0.16%) for those 65 and older [73], and over age 70 has increased to 270 in 6004 (4.5%) [100].

Failure of the pacemaker function is classified into levels of incompetence. Sinus node arrest is prolonged asystole leading to significant pauses between atrial beats where no AP is generated by the SA node. When asystole occurs, an escape rhythm AP may occur. Sinoatrial exit block describes impairment of the AP from leaving the SA node and is classified by degrees the impairment. With first degree block, the AP conduction is delayed leading to sinus bradycardia. There are two types of second degree block which prevent AP exit at a periodic interval. With Type 1 second degree block the time between two adjacent P waves, the P-P interval, gradually shortens on each subsequent beat until a beat is lost, the P-P interval returns to its full duration and the pattern repeats. Type 2 is similar to Type 1 except the P-P interval remains constant until a beat is lost. With third

degree block multiple consecutive APs are generated within the SA node but blocked from exiting to the conduction system [66].

Atrioventricular Node Block. AV block prevents sinus AP from propagating from the atrium into the ventricle. The block may occur in the AV node or the conductive His bundle that propagates the AP through the ventricle. There are three types of AV block.

First-Degree AV block is a prolongation of the P-R interval to a value greater than 200ms. The P-R interval remains constant and all P waves are followed by a QRS complex. The delay may originate in the AV node or the His bundle [38, 151].

Much like second-degree SND, second-degree AV block results in loss of QRS complexes with a constant periodicity. With type I second-degree AV block, also known as Mobitz I or Wenchenbach, the P-R interval grows progressively longer until a QRS is lost. The interval then resets to its shortest value and the cycle repeats. The source of type I block is almost always located in the AV node though in rare cases it may originate in the His bundle. Type II AV block or Mobitz II describes a constant P-R interval that may be normal or prolonged with at least two conducted P waves before loss of a P wave. The periodicity of the dropped P wave will hold constant and is given as a ratio (e.g. 2:1, 3:2 block). Type II block originates in the His bundle except for 2:1 which is a variant of Mobitz I that originates in the AV nodeI [38, 151].

Third-degree AV block is also known as complete heart block (CHB). All AP conduction is blocked. Most patients with third-degree block are sustained with escape rhythm. The ECG of such patients shows complete dysynchrony between P waves and elongated QRS complexes with P waves usually occurring at a higher rate than the QRS complexes. Block may occur in the AV node or the His bundle [38, 151].

Bundle Branch Block. The His bundle bifurcates into the right and left bundle branches providing the conduction pathways for each of the ventricles. The left bundle further bifurcates into the anterior fascicle and posterior fascicle. Bundle branch block (BBB) may occur in any of these branches and is described as unifascicular, bifascicular, or trifascicular disease depending on the

number of branches blocked [101]. When a block occurs, AP propagation through the remainder of the conduction pathway for the associated ventricle is disrupted.

The right and left bundle branches are isolated via the intraventricular septum. An AP can slowly conduct between the two branches but in a healthy patient the AP is still conducting through the higher speed branch cells leaving the conduction path cells in refractory by the time AP from the opposing branch has propagated through the septum. Thus, the slow path AP does not cause additional depolarization. When a branch is blocked however, this slower conduction path now sources the AP into the blocked path causing dysynchrony between the ventricles due to the increased propagation delay of the AP to the blocked ventricle. Left BBB is common for heart failure patients [38, 40].

Chronotropic Incompetence. Chronotropic incompetence (CI) is primarily an age related form of bradycardia where the patient's maximum heart rate is insufficient to support exertion [25]. It is not a defect of the SA or AV nodes but a gradual, age related decrease in β-adrenergic [135] responsiveness of the sympathetic nervous system [34], the body's signaling system for heart rate change. β-adrenergic [135] responsiveness is the primarily signaling associated with the fight or flight response. Heart failure patients administered β-blocker medication may also exhibit CI [161]

3.2.1.2 Tachycardia

Tachycardia describes conditions where the heart rate accelerates to a high rate without an external stimulus such as physical activity or adrenaline and may occur in any chamber of the heart. The clinical consensus heart rate for tachycardia diagnosis is 100 beats per minute and above [131]. Tachycardia is not always a sign of heart disease. Sinus tachycardia, where the rate is being set by the SA node in response to exercise, is normal, but resting sinus tachycardia above the 100 beat threshold is considered symptomatic.

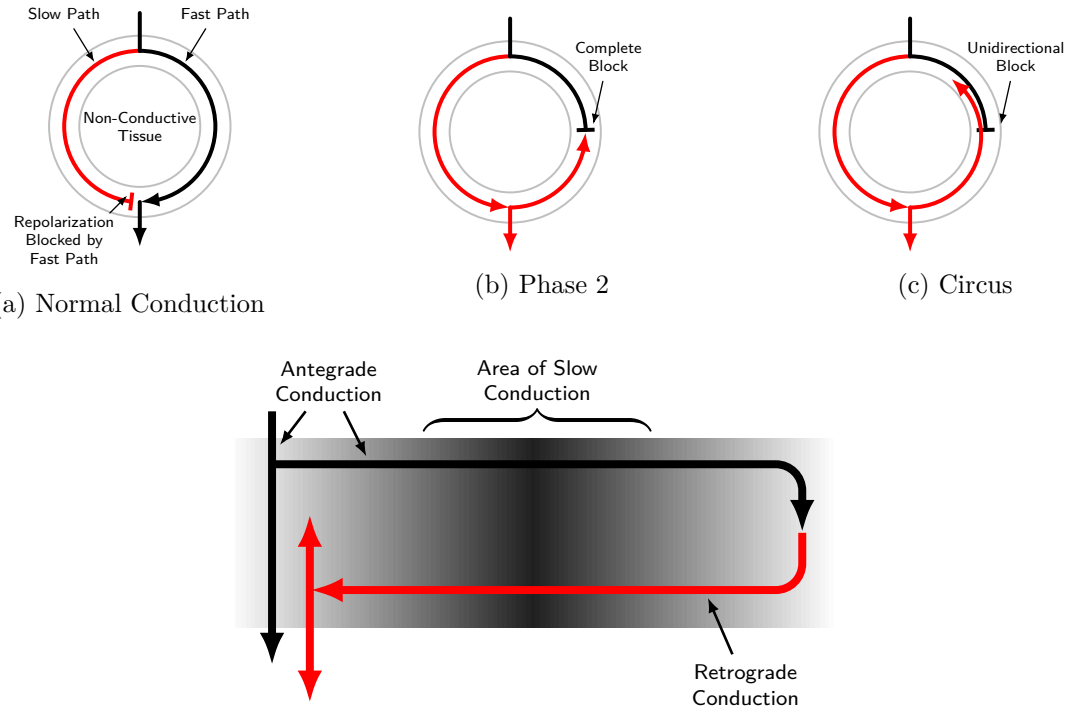
As a cardiac disease, tachycardia typically presents as flutter or fibrillation in the atria, ventricles, or both. Fibrillation is a life threatening arrhythmia where the atria or ventricles beat non-rhythmically due to chaotic electrical activity [126]. The affected chamber may cease pumping

blood. Flutter is another high rate arrhythmia where the atria or ventricles beat at a rapid and regular rate greater than 240 beats per minute [37, 126]. The ECG waveform for flutter will typically show a sine wave-like morphology. Patients suffering from flutter can be asymptomatic or may notice heart palpations.

Where bradycardia is a disease of AP loss, tachycardia is a disease of rapid AP propagation. In the right half of Figure 3.3 tachycardia arrhythmias are classified by their manifestation related to how the rapid AP are generated. Abnormal AP formation describes when alternate pacemaker cells begin spontaneously generating AP asynchronous to the SA node. Reentrant conduction describes multiple alternate pathways where a propagating AP can re-stimulate myocytes after they've completed refractory in a self sustaining loop.

Abnormal AP Formation. Pacemaker cells can be found anywhere in the conduction system from the SA Node down to the Purkinje fibers. They are identifiable by their lower (less negative) resting voltage and narrower AP waveform. The lower resting voltage gives them the propensity to self depolarize, starting an AP propagation. What keeps healthy heart from devolving into chaos is the rate of self discharge decreases for pacemaker cells further away from the SA. Because the SA node self discharges first, its AP propagation reaches and resets all slower pacemaker cells further down the conduction pathways [145]. This phenomenon explains why a patient with CHB or BBB can still have ventricular activity when the SA node AP is blocked. The slower pacemaker cells further down the conduction pathway take over, though at a slower rate.

Abnormal automaticity is a breakdown of this order where other cells begin to self depolarize at a rate faster than AP from the SA Node has propagated to them. The underlaying cause of this disease is not fully understood. The errant cells may be other pacemaker cells or traditional contractile cells that do not normally function as a pacemaker. Experimental results appear to point to a malfunction of the cell timing currents such that they modify the AP morphology of the cell and relaxing the self depolarization threshold [11, 13, 145]. Premature Ventricular Complexes (PVCs) are normally benign versions of this phenomenon [99]



(a) Normal Conduction

(b) Phase 2

(c) Circus

(d) Reflective

Figure 3.4: Conduction Paths

Triggered Activity occurs when the ionic flows of Ca and Na^+ during the cardiac cycle become unbalanced and the cell spontaneously begins depolarization. Early afterdepolarization (EAD) occurs during the phase 2 or phase 3 (plateau and repolarization) portions of the AP waveform [11, 13, 145]. This is sometimes referred to as R-on-T due to the ECG morphology where the new ventricular contraction R wave is seen riding on the refractory T wave from the previous contraction [118]. Delayed afterdepolarization (DAD) occurs when the cell spontaneously depolarizes during phase 4 due to an overload of Ca. DAD is a typical cause for superventricular tachycardia and atrial fibrillation [11, 13]

Reentrant Conduction. Reentrant conduction is categorized into three forms: circus, reflection, and Phase 2 [11, 76] and are typically created when a slower, alternate pathway allows for an AP to return to already contracted myocytes after they've completed phase 3 refractory and repeating the AP conduction through the original pathway. In a healthy individual, shown in

Figure 3.4a, the normal pathway (black arrow) beats the slower pathway (red arrow) to the circuit merging location. The faster pathway's AP propagated into the slower pathways exit, inducing normal depolarization and blocking the slower path's AP conduction. Figure 3.4b shows Phase 2 retrograde conduction. Here the faster path has failed and is bidirectionally blocked. The slower pathway becomes the method of AP conduction, splitting when reaching the exit point to continue down the original faster pathway while also moving backwards through the remainder of the blocked faster conduction pathway until reaching the blocked location. Figure 3.4c depicts Circus reentry, a form of fibrillation, where the fast path block is unidirectional and only allows conduction opposite the normal fast path flow. In this condition, the slower path AP propagation moves retrograde through the fast path and the block returning to the slow path starting point, restarting the slower pathway again forming a continual loop [11, 13].

Reflection conduction is shown in Figure 3.4d. While this form of reentry is still an open research question [13], Rozanski [121] demonstrated the pathway in experimental work. In reflection reentry, there is an area of slow conduction between two locations. The left end in the figure is the origin of the AP propagation. The left-to-right propagation is slowed while passing through the middle area before reaching the right side. The right side depolarizes and the AP begins to propagate back through the middle area. When it reaches the left side, the original conduction pathway has already exited refractory and, thus, is retriggered beginning another AP propagation. This form of reentry differs from circus as the AP propagation path is 'reflected' back through the same tissue from which it came.

Phase 2 reentrancy is so named due to distortion of the phase 2 portion of the AP waveform in affected cells [11, 13]. This form of reentry does not require special pathways but is a focal reentrancy caused by individual cells with abnormal phase 2 waveforms. These distorted waveforms can cause adjacent repolarized normal cells to depolarize and begin a new cycle of AP propagation.

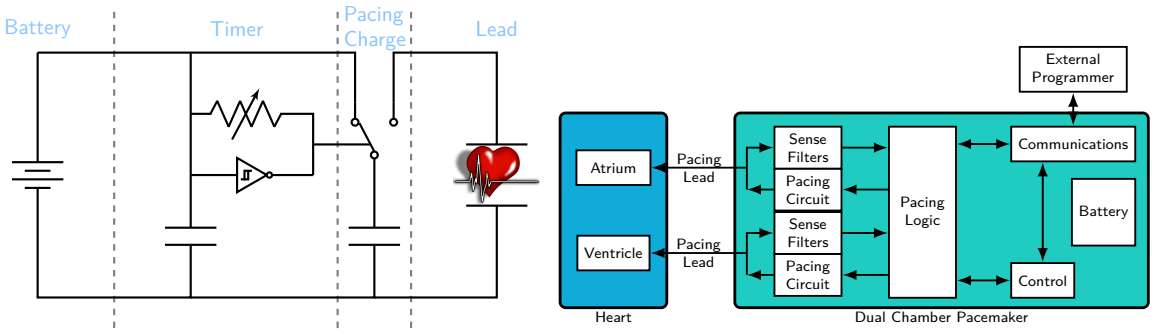
3.3 Pacemakers - A Brief Overview

When the heart's conduction system fails, coordinated contraction of the heart is impaired and the patient's health is compromised. A patient suffering from acute conduction blockage or interruption due to drugs, viral infection, or recovery from surgery leaving the heart weak may receive a temporary external pacemaker to provide support while the heart recovers. Chronic cardiac disease sufferers may be prescribed a Cardiac Implantable Electronic Device (CIED). CIEDs describes a category of implantable cardiac devices including bradycardia pacemakers, implantable cardiac defibrillators (ICDs) for tachycardia, cardiac resynchrony therapy (CRT) devices for heart failure patients, and ECG loop recorders for symptom diagnostics.

A pacing system consists of a pacemaker, colloquially known as 'the can', and wires (leads or pacing leads) to connect the pacemaker output to the myocardium. Pacemakers today are composed of a two part titanium shell containing the battery and electronics along with a connector or header block where the pacing leads attach. They are commonly implanted in a subclavical pocket under the skin of the left shoulder. From the pocket a local vein, typically the left Cephalic, is accessed for lead insertion into the right atrium [16]. The right atrial lead is secured near the SA node while the right ventricular lead continues through the tricuspid valve and is typically secured in the ventricle apex. Leadless pacemakers are a more recent development. These devices are significantly smaller and are affixed directly to the myocardium in the same location leads would be affixed for the larger devices.

3.3.1 Pacemaker Design.

The simplest pacemaker, a single chamber pace-only device, requires nothing more than a battery, switch, capacitor, and a timing circuit as shown in Figure 3.5a. In operation, the capacitor is connected to the battery through the switch to charge. When the timing circuit reaches the end of the interval between pacing pulses, the switch toggles to discharge the capacitor into the heart and then returns to charging the capacitor. If sufficient charge is delivered to the heart, cells near



(a) Simple Single Chamber Pacemaker (b) Dual Chamber Pacing System

Figure 3.5: Pacemaker System Designs

the distal tip of the pacing lead will depolarize and begin an AP propagation. The pacemaker has 'captured' the heart. Such a pacemaker will continually pace the heart at the rate set by the adjustable timer while ignoring intrinsic heart activity.

Modern pacemakers are significantly more complicated than the simple version in Figure 3.5a. Pacemakers today are composed of pacing logic, where the pacing therapies are encoded, pacing and sensing circuits for each chamber, control logic, and communications for changing therapy non-invasively. Figure 3.5b shows a basic block diagram.

As shown in Figure 3.5b, pacing leads serve dual purposes delivering the pacing pulse from the pacemaker to the myocardium while also routing EGM from the heart to the pacemaker's sense filters to enabling sensing of intrinsic heart activity. The sense filters are a combination of analog and digital filters with the purpose of attenuating noise while amplifying the intrinsic heart activity. P and R wave primary frequency composition is between 1 and 200Hz with the majority of spectral intensity between 1 and 80 Hz [112, 140]. At the lower end of this band is the patient's breathing ($\sim$0.5-1 Hz) while at the higher end is noise from muscle activation potential ($>$ 80 Hz) [140]. The pacemaker must be able to filter such body noise from the EGM to assure appropriate pacing. Additional filtering is typically required for removal of external interference that can fall into this frequency range. Most notable are power systems operating at 16, 50, and 60 Hz.

Upon completion of the filtering, the resultant signal is compared to a programmable threshold defining the minimum amplitude for the signal to be considered an intrinsic cardiac pace event. When the incoming signal from the lead exceeds this threshold an intrinsic event flag is generated. Programming this threshold value too high leads to intermittent missed events or complete inability to detect the intrinsic cardiac activity. Many modern pacemakers will periodically update the threshold value by performing a threshold test consisting of titrating the threshold value until loss of capture and then selecting a new permanent value set to this loss value plus a safety margin. An incorrectly programmed sensing threshold that causes unnecessary pacing will decrease the longevity of the implanted device.

Battery longevity is an important pacemaker feature and thus it is advantages to decrease the amount of pacing provided by the pacemaker, allowing the heart to intrinsically beat when possible to extend the time between device replacement. This leads to improved patient safety as surgical replacement of an implanted medical device carries roughly a 4% risk of surgical site infection [18]. While a larger battery can reduce this risk by increasing the time between needed replacements, battery size is limited for practical and cosmetic reasons.

The pacing logic block of Figure 3.5b contains the therapy algorithms. This logic monitors the output of the sensing filters and decides when to pace. These algorithms are manufacturer proprietary designs for unique market advantage. Ongoing research into the onset of tachycardia has provided insight into methods of detecting the onset of tachycardia and flutter arrhythmias leading to special algorithms that allow pacemakers to terminate the arrhythmia using antitachycardia pacing (ATP)s [45] decreasing the need to provide a defibrillation shock. Basic pacing logic will be presented in detail later in this section.

Once the pacemaker is implanted, there's no non-invasive way to update or change the patient's therapy. The communications block of Figure 3.5b represents the various H and E-field radios in use today. The clinician uses an external computer known as a programmer to view and change pacemaker parameters. Until recently, these communication channels were proprietary to each pacemaker manufacturer and incompatible between manufacturers. Newer pacemakers in-

corporate a Bluetooth radio allowing communication directly with patient cell phones though the telemetry protocols are still manufacturer proprietary.

Finally, the control block contains logic for all non-pacing related tasks including the operating system.

3.3.2 Modes of Pacing

Pacemakers come in a variety of pacing and sensing options. The North American Society of Pacing and Electrophysiology (NASPE) maintains Table 3.1 [17] that describes the primary features of any pacemaker. The NASPE system breaks down the features into five primary categories: Chambers Paced, Chambers Sensed, Response to Sensing, Rate Modulation, and Multisite Pacing. A pacemaker code must include the first three categories. The remaining two are optional and assumed to be code 'O' (none) if not present.

The most basic pacemaker is one that only provides pacing support in the ventricle with no sensing and synchronization of intrinsic heart activity. The NASPE code for this device is VOO. A pacemaker that adds sensing and synchronization in the ventricle has the code VVI for pace in the ventricle (V), sense intrinsic cardiac activity in the ventricle (V), and inhibit (I) a scheduled pace if a valid intrinsic ventricular event occurs.

Dual chamber devices typically can pace and sense in each chamber resulting in a coding of DDD or DDDR. The first two symbols define pacing and sensing in both the right atrium and ventricle. The next D defines that paces may be inhibited (I) or triggered (T). In this configuration, like the VVI device, a valid, sensed cardiac event can inhibit (I) the next scheduled pacemaker output or A sensed event that is deemed valid in all manner except its placement in time may trigger (T) a pace to be generated by the pacemaker upon completion of a timing sequence. The final R symbol, if present, indicates the pacemaker will operate in a rate responsive mode, denoting functionality that increases and decreases the pacing rate based on patient activity.

The NASPE code is used both to define the maximal capabilities of a pacemaker and the present settings of a pacemaker that is in use. Thus, a dual chamber DDDR pacemaker that can pace

I **Chamber(s) Paced**	II **Chamber(s) Sensed**	III **Response to Sensing**	IV **Rate Modulation**	V **Multisite Pacing**
O = None	**O** = None	**O** = None	**O** = None	**O** = None
A = Atrium	**A** = Atrium	**T** = Triggered	**R** = Rate Modulation	**A** = Atrium
V = Ventricle	**V** = Ventricle	**I** = Inhibited		**V** = Ventricle
D = Dual (A + V)	**D** = Dual (A + V)	**D** = Dual (I + T)		**D** = Dual (A + V)

Table 3.1: NASPE/BPEG generic codes for pacemakers.

and sense in both the atrium and ventricle may be programmed to DVI for a particular patient based on their disease state. In this case, the patient may have a healthy SA node but a faulty AV node where the atrium continues to beat on its own but the heart depolarization does not reliably reach the ventricle. Therefore, the pacemaker will monitor the atrial activity. When an atrial intrinsic pace is detected, a delay timer will be started. Should this delay expire with no valid ventricle intrinsic activity, the ventricle will be paced by the pacemaker. If intrinsic ventricular activity is detected and it falls within an allowable window during the delay, the scheduled ventricular pace will be inhibited.

3.4 Pacing Therapy

Pacemakers are cyclic devices that operate with a defined periodicity. The pacemaker in Figure 3.5a has very few requirements while dual chamber pacemakers with sensing and rate response have significantly more with complex interactions between them. This section will take a stepwise approach to presenting pacing requirements and nomenclature starting with the most basic pacemaker and building to more complex devices. The basic functionality each mode of pacing will be represented as an automaton and the requirements illustrated using ladder diagrams.

Figure 3.6 is an example diagram showing an ECG trace above time aligned with a labeled ladder diagram below. This ECG depicts two complete heart cycles of atrial and ventricular beats. The first atrial and ventricular beats were paced and can be recognized by their sharp positive

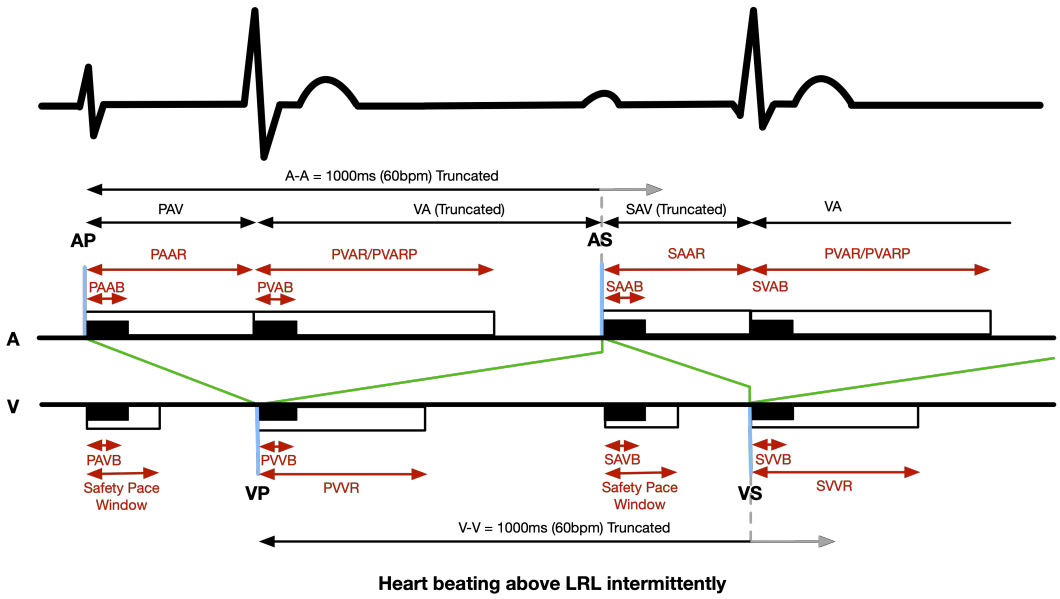


Figure 3.6: Example Ladder Diagram

and negative transitions. The second atrial and ventricular beats were intrinsic and show a similar waveform to Figure 3.2.

The A line of the ladder diagram depicted below the ECG trace represents activity in the atrium annotated above the line where events are vertical lines, timed periods are boxes, and horizontal lines represent intervals. Likewise, the V line with annotations below it represents events, periods, and intervals specific to the ventricle. Event designations are found in Table 3.2. Tradition calls timed sequences from a paced or sensed event in a chamber to a subsequent paced or sensed event an interval. All other timing sequences are referred to as periods. This example diagram shows a quite complicated sequence of interactions for a dual chamber pacemaker that will be described with requirements later in this section. The complexity does allow introduction of the acronyms and terminology that will used in the remainder of this document.

The boxes of the ladder diagram represent time periods and they have a unique acronym that describes what each represents. Table 3.3 is the secret decoder ring for the acronyms. For example PAAB stands for a paced atrial atrial blanking period or in other words the blanking period in the atrium after the atrium was paced by the pacemaker. Likewise, SVVR is the ventricular refractory

period following a sensed intrinsic ventricular beat. One exception to this nomenclature is the Paced Ventricular Atrial Refractory period (PVAR), which by tradition is typically annotated as PVARP.

Figure 3.6 also depicts the four intervals used in pacing. The A-A Interval represents the time between two atrial beats, intrinsic or paced. The V-V Interval represents the time between to ventricular beats. The AV Interval is the time from an atrial beat to the next ventricular beat. And the AV Interval which is the time from a ventricular beat to the next atrial beat.

The final element of the ladder diagram are the green lines moving between the atrial and ventricular timelines. These lines represent the elapsed time of the AV and VA intervals for dual chamber devices. For single chamber devices they represent the elapsed time for the A-A or V-V interval depending on pacemaker type. The x-axis represents the time elapsed for the green line while the y-axis represents percentage complete. As an example, when a green line starts from a ventricular event and begins moving upward, the interval is complete when it reaches the atrial line. An interval can be truncated by an event such as the atrial sensed event (AS) shown in the diagram. A truncated interval is depicted by the line terminating with a vertical line. The depiction is abused slightly for single chamber devices as the other chamber isn't implemented as seen in the VOO pacemaker diagram of Figure 3.7a.

Pacemakers that do not sense the intrinsic heart activity are a simple model for introducing pacing and the time intervals between them. Periods are much more complex as they can be reset or

Code	Event
AP	Atrial Pace
AS	Atrial Sensed intrinsic beat
VP	Ventricular Pace
VS	Ventricular Sensed intrinsic beat
PVC	Premature Ventricular Contraction
PVC-R	Premature Ventricular Contraction Refractory
AR	Atrial Refractory intrinsic beat
VR	Ventricular Refractory intrinsic beat

Table 3.2: Event Labels

x---	-x--	--x-	---x
Event	Event Chamber	Interval Chamber	Interval
P - Pace S - Sense	A - Atrium V - Ventricle	A - Atrium V - Ventricle	B - Blanking R - Refractory

Table 3.3: Construction of timing interval acronyms

truncated due to intrinsic activity. They will be introduced after the pace only modes of operation.

VOO Requirements. The simplest pacemakers are those that simply pace the heart at a regular rate such as a VOO device. As the device acronym specifies, it must only pace in the ventricle and can be represented by the simple timed automaton shown in Figure 3.7b assuming a rate of 1000ms or 60 bpm and a 1000Hz clock rate. Each time the Lower Rate Interval counter t reaches 1000, a pace is generated and the counter is reset. Figure 3.5a is an example implementation.

The first requirement for a VOO device is to only pace the ventricle. While this requirement may seem odd, dual chamber pacemakers can be programmed to single chamber modes. In such cases, the pacemaker must not pace in the off chamber.

VOO-1

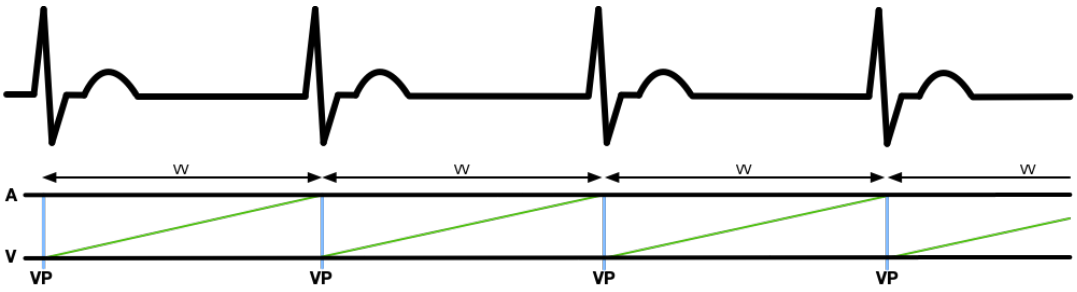
The pacemaker shall only pace the ventricle.

The Lower Rate Interval (LRI) or Lower Rate (LR) is the absolute slowest the pacemaker will allow the heart to beat. Because a VOO device is not sensing intrinsic activity, this is the rate it will pace at.

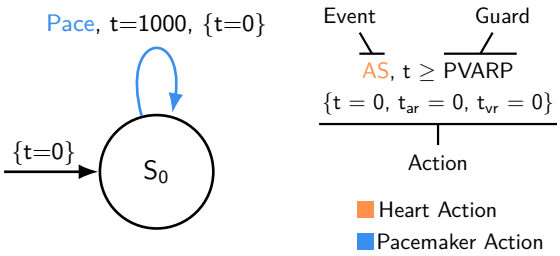
VOO-2

The pacemaker shall pace the ventricle at the Lower Rate Interval (LRI).

The ladder diagram in Figure 3.7a shows the VOO pacemaker meeting these requirements. There are no AP markers and a VP occurs at the end of each V-V interval as this interval is



(a) VOO Ladder Diagram



(b) VOO/AOO Automaton

Figure 3.7: VOO Automaton and Ladder Diagrams

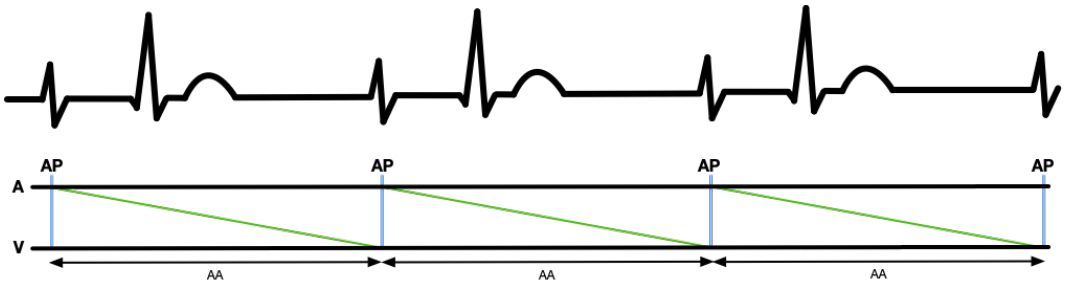


Figure 3.8: AOO Ladder Diagram

equivalent to LRI for a pace only device. The ECG trace in the figure has been simplified to not show intrinsic atrial activity as it is irrelevant for VOO pacing.

AOO Requirements. AOO pacing is identical to VOO pacing and a single chamber device can usually be configured to do either with only a change in the output voltage and placement of the lead in the paced chamber. Figure 3.8 shows the AOO ladder diagram while the pacing automaton is exactly the same as VOO shown in Figure 3.7b.

AOO-1

The pacemaker shall only pace the atrium.

AOO-2

The pacemaker shall pace the atrium at the Lower Rate Interval (LRI).

The ECG in Figure 3.8 depicts a patient with no sinus rhythm and full AV conduction so intrinsic ventricular beats follow each atrial pace.

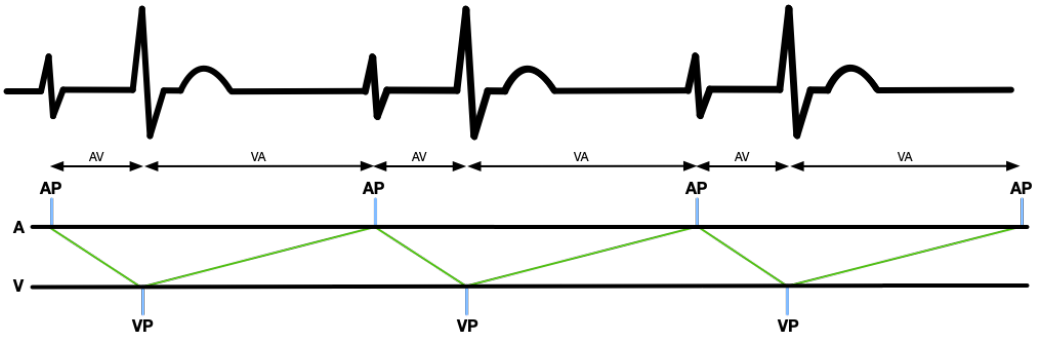
DOO Requirements. DOO pacing breaks the LRI into two intervals of unequal length. The AV interval runs from the atrial pace to the ventricular pace and is represented as state S_0 in the DOO automaton of Figure 3.9b. The continuously incrementing counter t measuring the length of the AV interval can be seen as the green line in the Figure 3.9a ramping from the AP to VP annotations. When the counter reaches the AV interval length, a ventricular pace is generated, the counter reset and the automaton transitions to state S_1 for the VA interval.

The VA interval runs from the ventricular pace to the next atrial pace, depicted in Figure 3.9a with the green line ramping up from the V line to the A line and ending with the AP marker. When the counter reaches the length of the VA interval, an atrial pace is generated, the counter reset and the automaton transitions back to state S_0. The VA interval is about three times longer than the AV interval and varies by patient. The automaton is depicted for a 1000ms LRI, an AV interval of 250ms, and a VA interval of 750ms.

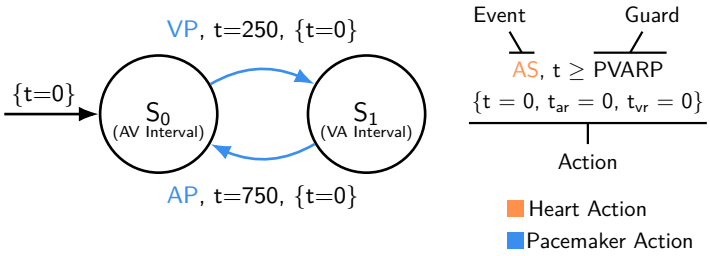
The DOO requirements are similar to AOO and VOO except now time is referenced to the AV and VA intervals.

DOO-1

The pacemaker shall pace the atrium at the end of the VA Interval.



(a) DOO Ladder Diagram



(b) DOO Automaton

Figure 3.9: DOO Automaton and Ladder Diagrams

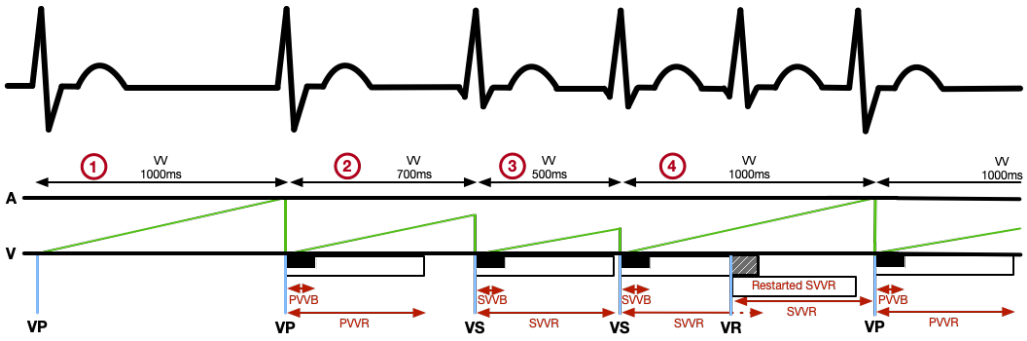
DOO-2

The pacemaker shall pace the ventricle at the end of the AV Interval.

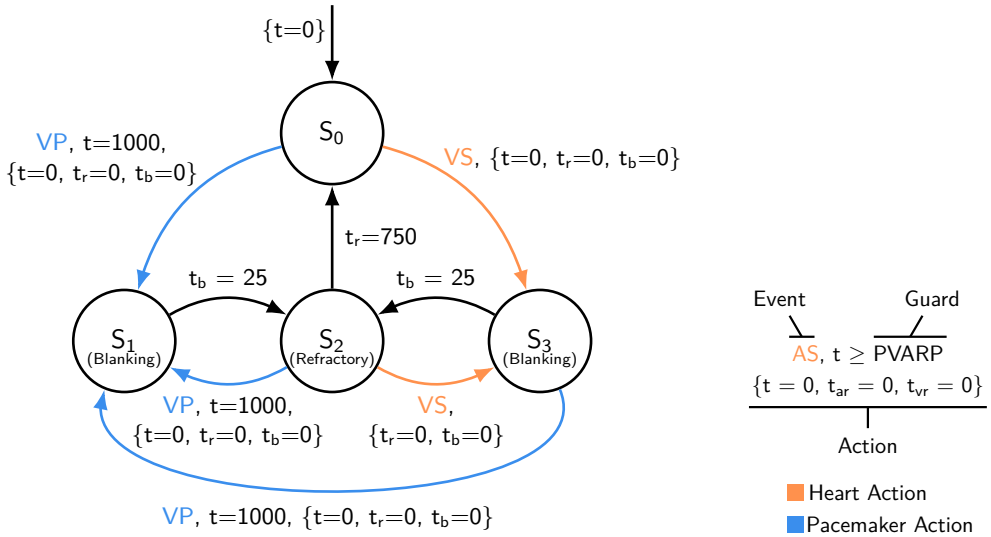
DOO-3

The time from atrial pace to next atrial pace shall be the Lower Rate Interval (LRI).

VVI Requirements. VVI pacing introduces intrinsic sensing, blanking and refractory periods, and preemption of scheduled pacing due to heart activity. A goal of modern pacing is to allow the heart to intrinsically sustain the patient when possible. Thus, sensing intrinsic beats is an integral part of the pacing algorithm. At the start of each V-V interval of a VVI pacemaker, the next ventricular pace is scheduled to occur at the end of the interval. When an intrinsic beat is detected, the pacemaker must decide its validity and, if valid, inhibit the next scheduled pace. The V-V interval then restarts. Beat validation will be defined through the following discussion of



(a) VVI Ladder Diagram



(b) VVI Automaton

Figure 3.10: VVI Automaton and Ladder Diagrams

the VVI ladder diagram.

The first four V-V intervals have been numbered in Figure 3.10a for ease of introducing each of these concepts. For this example the V-V interval (t) is 1000ms (60bpm), blanking period (t_b) is 25ms, and the refractory period (t_r) is 350ms as shown in Figure 3.10b.

(1) **Pacemaker startup.** This segment depicts the start up of the pacemaker. At this point the pacemaker has no history of previous intrinsic events and thus does not know where in the cardiac cycle the heart currently is and assumes a valid intrinsic event may occur at any time. This is state S_0 of the VVI automaton. The V-V interval timer t begins running (the green line rising from

the first VP label). The automaton remains in state S_0 until either t reaches the interval end or a sensed event occurs. In this case, no intrinsic occurs, the interval timer reaches maximum, a pace is generated, and the automaton transitions to state S_1.

VVI-1

The pacemaker shall pace the ventricle at the end of the V-V Interval.

VVI-2

The pacemaker shall only sense in the ventricle.

VVI-3

The pacemaker shall only pace in the ventricle.

VVI-4

Ventricular paces shall start a V-V Interval.

VVI-5

The V-V Interval shall be equivalent to the LRI.

VVI-6

On the first interval following poweron, the blanking and refractory periods shall be length 0.

(2) **Sensed Event 1.** The pacemaker is now synchronized with the cardiac cycle. In the ladder diagram a blanking period (black box) has been added and labeled PVVB to designate it as the ventricular blanking period following a ventricular pace. Paced and sensed events can saturate the pacemaker sensing amplifier circuit, leaving it unable to detect new intrinsic beats until it recovers. A short blanking period of 10-25ms is typically used for amplifier recovery where any detected

events are assumed to be noise and are ignored. The white box, labeled PVVR, is the ventricular refractory period following a ventricular pace. This period accounts for the time needed by the myocardial cells to complete Phases 0-3 of their action potential and return to resting state.

The automaton remains in state S_1 until the blanking period t_b expires and then transitions to state S_2 to time the remainder of the refractory period, t_r. In this example no intrinsic event occurs during the refractory period, t_r expires, and the automaton transitions to state S_0. While in State S_0 a sensed event occurs at 700ms after the VP. Since the blanking and refractory periods have expired but the V-V interval has not (green line did not reach the Atrial timeline), this is a valid intrinsic beat, annotated as VS in the ladder diagram. The V-V interval, blanking and refractory periods are reset and the automaton transitions to state S_3. The ventricular pace scheduled to occur at the end of the V-V interval has been inhibited by the intrinsic beat. This is the **I** of the VVI coding.

VVI-7

During the refractory period all ventricular sensed events shall be refractory senses

VVI-8

Non-refractory, sensed events shall start a blanking period.

VVI-9

Non-refractory, sensed events shall start a refractory period.

VVI-10

Non-refractory, sensed events shall inhibit the next scheduled ventricular pace.

VVI-11

Non-refractory, sensed events shall restart LRI.

VVI-12

Paced events shall start a refractory period.

VVI-13

Paced events shall start a blanking period.

③ **Sensed Event 2.** This interval begins in state S_3 after the VS that ended the previous interval. The periods are labeled accordingly as SVVB and SVVR for the post sensed ventricular blanking and refractory periods. Again, like the previous interval an intrinsic sensed event occurs. This time at a time of 500ms after the previous VS. As the refractory period ended at 350ms, this is still a valid VS and the interval ends.The automaton trace for this interval would be $S_3 \longrightarrow S_2 \longrightarrow S_0 \longrightarrow S_3$.

④ **Refractory Sensed Event.** Similar to the previous interval, in this interval the automaton starts in state S_3 after the VS. Again, the blanking period expires and the automaton transitions to the refractory state S_2. This time however, a VS is detected before the refractory period (SVVR) has expired. When an input to the sense amplifier surpasses the detection level during the refractory period and outside the blanking period, it may be noise, retrograde or reentrant conduction, or a valid beat. In all cases, because the refractory period has not completed, the pacemaker assumes the myocardium has not yet returned to its resting state and subsequently is not ready for the next contraction. For this reason, the VS does not end the interval.

The sensed event is labeled as a refractory ventricular beat (VR). Occurring during the refractory period, the myocardium may not have sufficiently returned to a resting state and this event may not have been an effective beat. The event may have, however, caused some portion of the myocardium to again depolarize. The pacemaker recognizes this event as refractory but insufficient to end the interval. The automaton transitions $S_2 \longrightarrow S_3$, restarting the refractory and blanking periods. Each subsequent VR will continue to restart these periods until either a non-

refractory sense occurs ending the interval or the interval time expires and the pacemaker paces. An example trace where a non-refractory sense ends the interval might be $\{S_3 \longrightarrow S_2 \longrightarrow S_3 \longrightarrow S_2 \longrightarrow S_0 \longrightarrow S_3\}$ Example traces of the interval ending with a paced event are: $\{S_3 \longrightarrow S_2 \longrightarrow S_3 \longrightarrow S_2 \longrightarrow S_1\}$, $\{S_3 \longrightarrow S_2 \longrightarrow S_3 \longrightarrow S_2 \longrightarrow S_0 \longrightarrow S_1\}$ or $\{S_3 \longrightarrow S_2 \longrightarrow S_3 \longrightarrow S_1\}$. In these final examples, the pacemaker will generate a pace at the end of the interval even though the refractory or blanking period has not expired. Doing otherwise would violate the requirement to pace at the end of the V-V interval.

VVI-14

Refractory, sensed events shall terminate the current refractory period and start sensed refractory and sensed blanking periods.

DDD Requirements. Building on the concepts and requirements of the single chamber VVI pacing mode and dual chamber DOO pacing mode, the DDD pacing mode adds additional complexity in order to maintain AV synchrony. DDD mode both senses and paces in the atrium and ventricle where sensed intrinsic events inhibit the next scheduled pace. The requirements presented in this section are subset of the requirements specified in the Boston Scientific pacemaker specification [86] and those implied in the Medtronic pacing leads analyzer [103] and Medtronic ADVISA™ pacemaker [102] clinician manuals.

To begin illustrating DDD requirements, Figure 3.11a depicts two cycles of the heart. In the first cycle, both chambers are paced. In the second cycle the heart intrinsically beats. Due to the increased complexity of dual chamber pacing, the figure has been augmented with the corresponding trace through the DDD automaton of Figure 3.11b above the ECG strip.

Figure 3.11a begins with an atrial pace starting the post atrial pace blanking and refractory periods PAAB, PAVB, PAAR, and PAVR. In most applications, the PAVR duration is set to zero. The DDD automaton is in state S_0.

DDD-1

Paced atrial events shall start an atrial blanking period.

DDD-2

Paced atrial events shall start a ventricular blanking period.

DDD-3

Paced atrial events shall start an atrial refractory period.

DDD-4

Paced atrial events shall start a ventricular refractory period.

DDD-5

Paced atrial events shall start a Paced AV Interval (PAV).

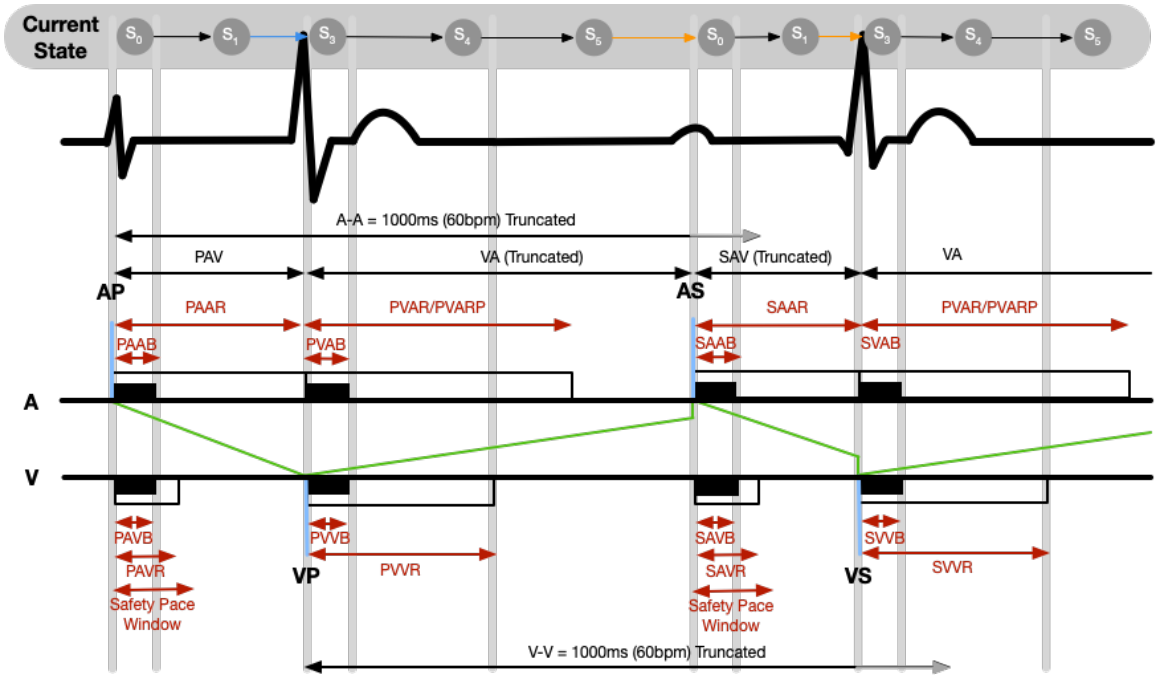
DDD-6

Paced atrial events shall schedule a ventricular pace to occur at the end of PAV.

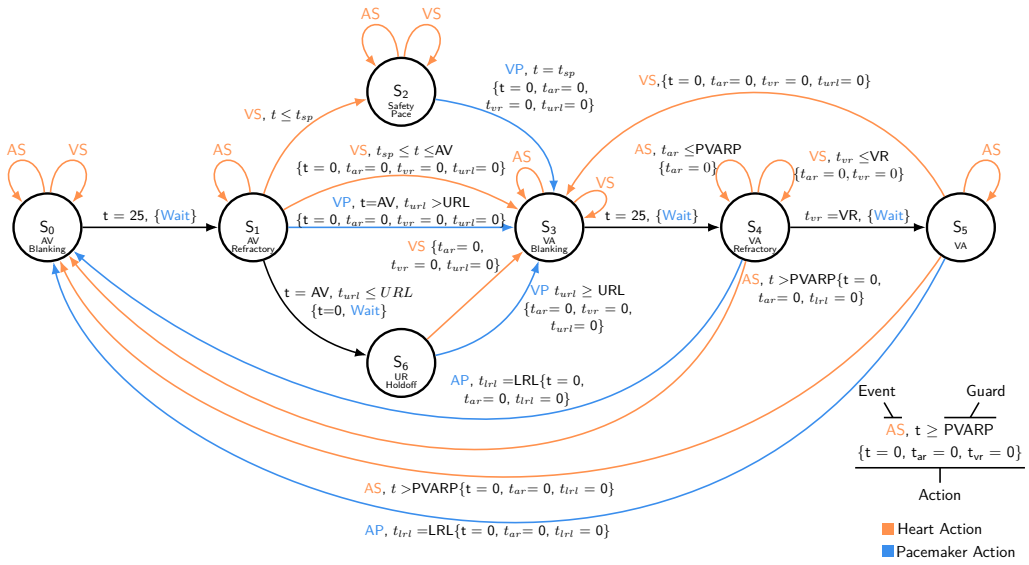
DDD-7

Sensed intrinsic events during a blanking period shall be ignored.

When blanking completes, the automaton transitions to state S_1 for the remainder of the PAV interval. The PAV interval of Figure 3.11a ends with no sensed intrinsic event occurring. The automaton transitions to state S_3,the scheduled ventricular pace (VP) occurs, the post ventricular pace blanking and refractory periods start and the VA interval begins.



(a) DDD Ladder Diagram



(b) DDD Automaton

Figure 3.11: DDD Automaton and Ladder Diagrams

DDD-8

Paced ventricular events shall start an atrial blanking period.

DDD-9

Paced ventricular events shall start a ventricular blanking period.

DDD-10

Paced ventricular events shall start an atrial refractory period.

DDD-11

Paced ventricular events shall start a ventricular refractory period.

DDD-12

Paced ventricular events shall start a Paced VA Interval (VA).

DDD-13

Paced ventricular events shall schedule an atrial pace to occur at the end of the VA interval.

Continuing on with Figure 3.11a, the post VP blanking period ends and the automaton transitions to the ventricular refractory period S_4. In this example, the ventricular refractory period expires with no intrinsic events and the automaton moves on to S_5 for the remainder of the VA interval. An atrial sensed event occurs before the interval expires. The atrial sense is outside of the atrial refractory period so this is a valid atrial sense, inhibiting the next scheduled atrial pace, terminating the current VA interval while starting a sensed AV (SAV) interval along with sensed atrial refractory and blanking periods and schedules the new ventricular pace to occur at the end of the SAV interval. The automaton returns to S_0.

DDD-14

Sensed atrial events shall start an atrial blanking period.

DDD-15

Sensed atrial events shall start a ventricular blanking period.

DDD-16

Sensed atrial events shall start an atrial refractory period.

DDD-17

Sensed atrial events shall start a ventricular refractory period.

DDD-18

Sensed atrial events outside of the atrial refractory period shall inhibit the scheduled atrial pace and start a Sensed AV Interval (SAV).

DDD-19

Sensed atrial events outside of the atrial refractory period shall schedule a ventricular pace to occur at the end of SAV.

Finally, the post AS blanking periods expire and the automaton transitions again to S_1 for the remainder of the SAV interval. Before the end of the interval, a valid ventricular intrinsic beat is sensed. This beat terminates the current SAV interval while starting post ventricular beat blanking and refractory periods, the VA interval, and scheduling of the next atrial pace. The automaton moves to S_3.

DDD-20

Sensed ventricular events shall start an atrial blanking period.

DDD-21

Sensed ventricular events shall start a ventricular blanking period.

DDD-22

Sensed ventricular events shall start an atrial refractory period.

DDD-23

Sensed ventricular events shall start a ventricular refractory period.

DDD-24

Sensed ventricular events shall start a Sensed VA Interval (VA).

DDD-25

Sensed ventricular events shall schedule an atrial pace to occur at the end of the VA interval.

The pacemaker presented here has two safety features, safety pacing and upper rate hold-off. Ventricular safety pacing (VSP) is a safety feature to prevent inappropriate inhibiting of a ventricular pace after an atrial pace in crosstalk conditions where the atrial pace is detected by the ventricular sense filter as a valid VP. Figure 3.12 shows an example of VSP in action. With the automaton beginning in state S_0, the post AP blanking periods expire leading to a transition to S_1. A VS is detected 75ms into the 150ms VSP window triggering the VSP feature and the automaton transitions to S_2 and waits for the VSP window to expire. The automaton then paces the ventricle and moves to S_3 as normal. No intrinsic, non-refractory senses occur allowing the automaton to transition to S_4 and then S_5 as the periods expire.

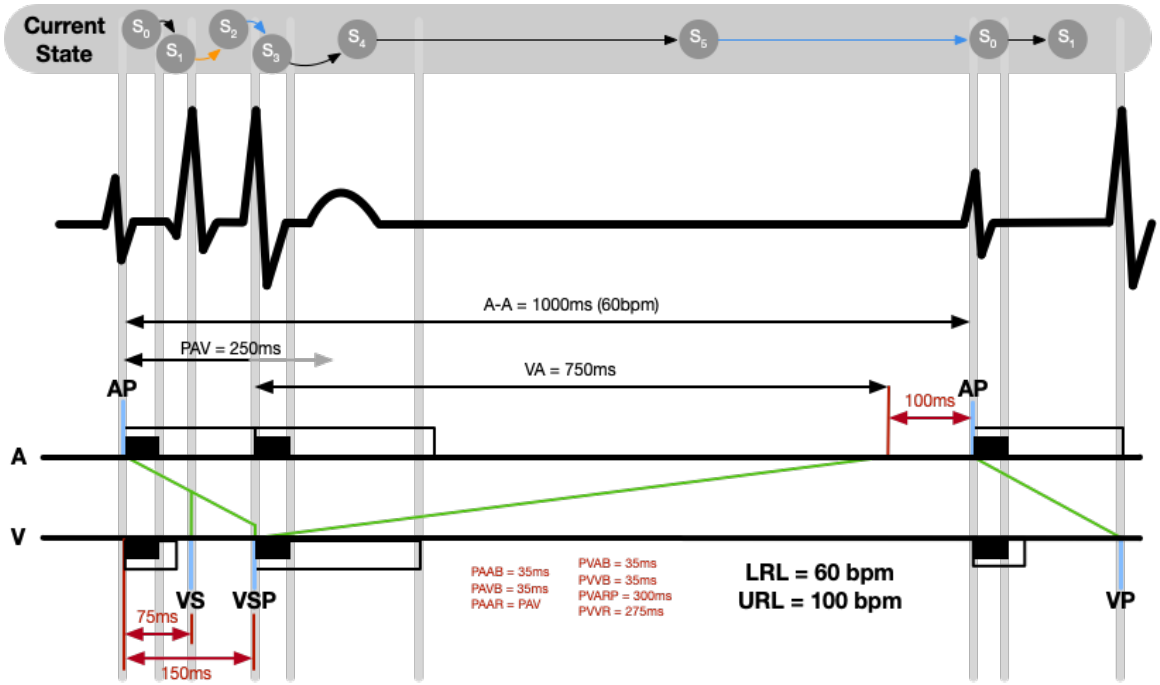


Figure 3.12: DDD Safety Pacing and VA Extension

DDD-26

Sensed ventricular events occurring inside the Ventricular Safety Pace window shall schedule a ventricular safety pace to occur at the end of the VSP window.

DDD-27

A Ventricular Safety Pace shall start a VA interval.

At this point, the VA interval will expire since no intrinsic atrial beats were detected. This should enable the scheduled atrial pace but to do so would violate the LRI, essentially pacing the patient at a higher rate than programmed. This violation can be determined by adding up the time from the previous atrial event. The LRI is set to 1000ms. The AV interval was the length of the VSP window, 150ms and the subsequent VA interval was its full length of 750ms for a total time of 900ms - 100ms short of the required LRI. To maintain the LRI from atrial event to the next atrial pace, state S_5 must extend the VA interval until the LRI has been satisfied at which point the VP

occurs and the automaton transitions back to S_0. A valid AS can always shorten the AA timing because the patient's heart is beating faster than the set lower rate. An AP must never do this.

DDD-28

When a VA interval expires before LRI is satisfied, the VA interval shall be extended until LRI satisfaction.

Upper Rate Holdoff is another feature to prevent the pacemaker from beating too quickly. Figure 3.13 is a somewhat contrived scenario designed to purposely invoke this feature by lowering the Upper Rate Limit (URL) to be close to LRI. The initial portion of this ladder diagram proceeds as normal ($\{S_0 \longrightarrow S_1 \longrightarrow S_3 \longrightarrow S_4\}$). At this point, a ventricular beat is detected. This ventricular beat is defined as a premature ventricular contraction (PVC) because it is not preceded by an AP or non-refractory AS and due to the ventricular refractory period still in effect, this is a refractory PVC (PVC-R). The PVC-R terminates the current PVVR and PVARP and starts a new SVVR and SVAR while looping the automaton back to S_4. An atrial sense occurs next. Since PVARP had expired, this is a valid, non-refractory atrial sense, terminating the VA interval, starting the SAV interval, and moving the automaton to S_0. The trace continues through the AV interval ($\{S_0 \longrightarrow S_1\}$) with no intrinsic events detected.

DDD-29

A non-refractory ventricular sensed event without a preceding atrial paced or non-refractory atrial sensed event shall start new SVAR and SVVR periods, restart the URL and the VA intervals, and inhibit the currently scheduled atrial pace and schedule a new atrial pace.

DDD-30

A refractory ventricular sensed event without a preceding atrial paced or non-refractory sensed event (PVC-R) shall start new SVAB, SVVB, SVAR and SVVR periods.

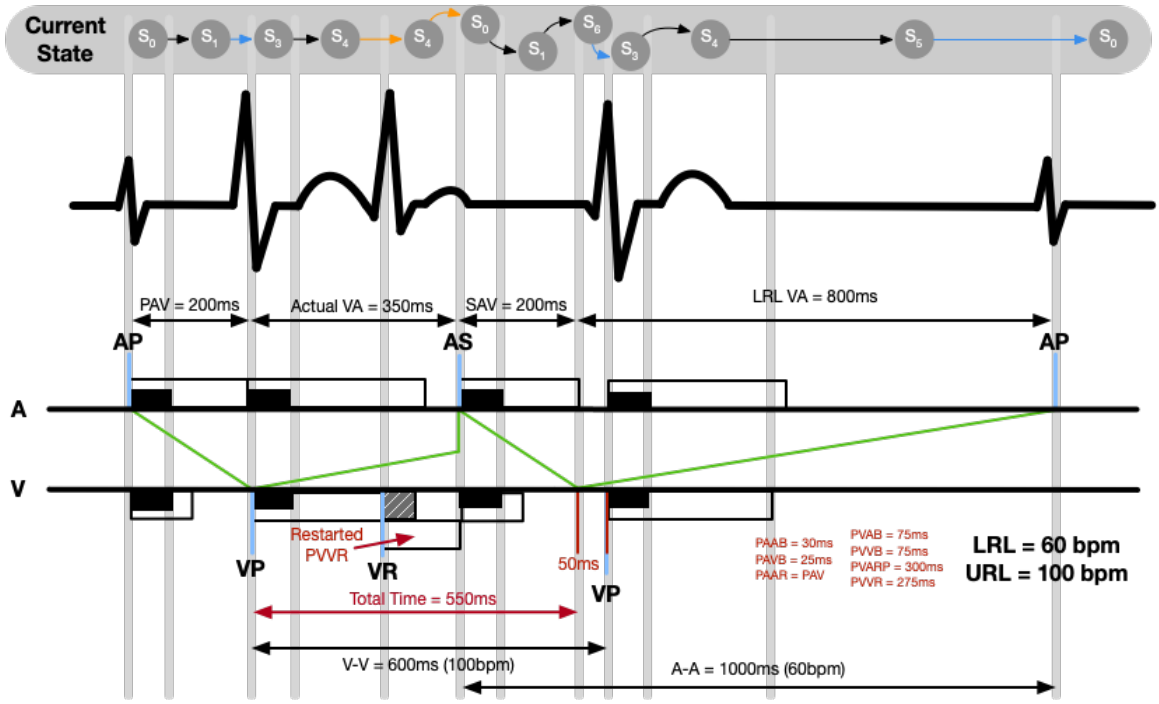


Figure 3.13: DDD URL Holdoff

DDD-31

Ventricular refractory period extension beyond the VA interval due to consecutive PVCs shall cause the atrium to be paced at the end of the VA interval (noise reversion)

The upper rate for this scenario is set to 600ms and is measured from the last ventricular event, paced or sensed to the next ventricular pace. In this figure, the total time passed was 350ms for the atrial sensed event truncated VA interval and 200ms for the uninterrupted AV interval for a total of 550ms. The ventricular pace must be held off for 50ms. Unfortunately, this delay will cause the next atrial pace to violate LRI by 50ms. The solution is for the automaton to transition to S_6. Here the new VA interval begins as normal so that LRI is not violated but the ventricular pace is held off for 50ms. Once the hold off expires, the ventricular pace occurs, the blanking and refractory period begin and the automaton transitions to S_3 as normal.

DDD-32

The VA interval shall always start at the end of the preceding SAV or PAV interval.

DDD-33

A scheduled ventricular pace shall be held off until the upper rate limit (URL) has been satisfied.

Chapter 4

Preliminaries

The increasing complexity of modern pacemaker is driving a commensurate increase in the risk of design errors escaping into implanted devices. As highlighted in Chapter 1, the current development process is ill equipped to detect or prevent errors from occurring, relying primarily on inspection, verification, and validation after the fact. Formal specifications and formal methods have been proposed to detect errors earlier in the design process [156, 113]. In this section, we review preliminaries on automata and logic based formalisms, AI technologies, and software design process used throughout this dissertation.

4.1 Automata and Logic Based Formalisms

Models and specifications are essential elements of the design process, providing the mechanism to communicate what a design is and how it should work. Specifications describe the intended design, typically as a list of statements or requirements representing the needed functionality including safety and performance [124]. A model is an abstraction of the intended design representing important design aspects with sufficient detail to clearly communicate intent. Models enable analysis of the design and may also represent the environment the design will interact with. The final product's quality and safety is predicated on the quality of the models and requirements used during the design process. They form the foundation of this thesis for integration of reinforcement learning into the design process to remove error prone process steps and enhance patient therapy.

The case studies presented later in this thesis use RL in various stages of the design process

to demonstrate how RL can replace parts of the design process where defects occur while gaining RL's ability to enable novel new product capabilities through creative exploration of the available design space. To do so, correct RL training requires an environmental model and a scalar rewards mechanism capable of giving positive and negative feedback based on the RL agent's actions. The remainder of this section introduces the modeling techniques and formal logics that will be used for the models and specifications in the case studies.

Formal logics are a way to reason about and test discrete relationships between objects using a formal language. Unlike natural languages, formal languages are restrictive and succinct allowing a practitioner to describe complex relationships without ambiguity. Formal language syntax constricts statements to grammatically well-formed sentences with a precise interpretation. With formal language statements, a design may be tested and verified through deductive theorem proving, behavioral counter-examples, check specification consistency, and be used as a test oracle [88].

One of the most basic formal logic is First Order Logic (FOL) which allows for negation, conjunctive, and disjunctive relationships along with quantification. Büchi [119]'s work with first order formal logic demonstrated specification satisfaction is decidable when the logic is converted to infinite word or ω-automaton. This realization has provided the foundation for model checking tool [14] and led to the development of algorithms for the conversion from logic to automaton [157, 148], a process used later in the case studies.

In addition to modeling systems, models can also be used as a graphical representation of formal language statements. To ease understanding of some formal language semantics presented in this section, automata based models will be used to illustrate the basic concepts, as was seen in already in Section 3.3. As such, modeling based on Markov Decision Processes and automata will be presented first followed by the formal logics used in the case studies.

4.1.1 Markov Decision Processes

In reinforcement learning, the Markov Decision Process (MDP) is a key element of the learning process. It provides a means to describe the available actions and impact of those actions in the

evolution of an environment. By collecting rewards from an MDP model for each action taken an RL agent can identify an optimal set of actions or policy for navigating the environment.

The MDP is a method for modeling decision processes in stochastic environments based on the Markov property, namely that the next model state is solely dependent on the current state and action. In other words, the MDP is memoryless and requires no knowledge of states previously visited. Given a set of states S and actions A, this memoryless property can be stated as the probability of the next state based on previous history where there is equivalence between considering only the current state and action to considering all states seen and actions taken so far:

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_0, a_0, s_1, a_1, \ldots, s_t, a_t).$$

Formally, a rewardful Markov decision process $\mathcal{M}$ is a tuple $(S, s_0, A, P, R, \gamma)$ where: S is a set of *states*, $s_0 \in S$ is the *initial state*, A is a set of *actions*, $P : S \times A \times S \to [0, 1]$ is a state transition probability function where $P_a(s', s) = P(s_t = s'|s_{t-1} = s, a_{t-1} = a)$, R is the expected reward function $R : S \times A$ where $R(s, a) \to E(R_t|s_{t-1} = s, a_{t-1} = a)$, and $\gamma \to [0, 1]$ is the reward discount factor. If both S and A are finite, the MDP is **finite** and, if no transition in P is stochastic, a finite MDP is equivalent to a **finite automaton**.

Using the definition of $\mathcal{M}$, interesting statements can be made about the evolution of $\mathcal{M}$ over a set of states and actions. For any state $s \in S$, let $A(s)$ denote the set of actions that can be selected in state s. A run r of $\mathcal{M}$ is a finite sequence $\langle s_0, a_0, s_1, a_1, \ldots, s_n\rangle \in (S \times A)^* \times S$ such that $P(s_{i+1}|s_i, a_i)>0$ for all $0 \leq i < n$ and $r \in \mathcal{R}^{\mathcal{M}}$ is a set of runs of the MDP. $\mathcal{R}^{\mathcal{M}}(s)$ then is the set of runs starting from state s, $last(r)$ represents the last state of a finite run r and $len(r)$ is the number of transitions in r.

A policy in $\mathcal{M}$ is a function $\pi : S \times A \to [0, 1]$ where $\pi(s, a) = \pi(s = s_t|a = a_t)$ providing a probabilistic mapping of the next action based on the current state. A policy is said to be deterministic if $\pi(s, a) \to [0, 1]$ for all $a \in A$ and $s \in S$. Let $\mathcal{R}_\pi^{\mathcal{M}}(s)$ denote the subset of runs in $\mathcal{R}^{\mathcal{M}}(s)$ that correspond to policy π with initial state s and $\Pi_{\mathcal{M}}$ be the set of all policies over $\mathcal{M}$.

This leads to two equations fundamental to reinforcement learning. These equations calculate

the expected return of the current policy based on runs through the environment. The RL agent uses this information to improve the policy with the goal of increasing the expected return of future runs. The expected return following policy π when starting in state s is the **value function** $v_\pi(s)$:

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right], \text{ for all } s \in S. \tag{4.1}$$

Similarly, the **action-value function** $q_\pi(s, a)$ provides the expected return of action state pairs:

$$q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]. \tag{4.2}$$

The expected return for each of these functions discounts future returns by the MDP discount factor γ, increasing the impact of near-term rewards.

4.1.2 Timed Automata

With real-time systems, the time between events is critical to properly responding to changes in the environment. Timed automata are a generalization of finite automata by adding mechanisms for modeling time-constrained evolution of systems [8]. In their work, Alur and Dill defined the concept of a timed language L over an alphabet Σ as a sequence of timed words over the nonnegative rational numbers where each timed word $(\sigma, t) \in L$ is composed of a symbol $\sigma \in \Sigma$ and the time of the symbol's occurrence t where t monotonically increases:

$$(\sigma_0, t_0), (\sigma_1, t_1), \ldots, (\sigma_i, t_i) \text{ where } t_i \leqslant t_{i+1} \text{ for all } i \geqslant 1, t_i \in R_{\geq 0}.$$

Removing the time stamps from a timed language L results in the $Untime(L)$ language which reduces the timed language to only contain the sequence of words over Σ sans time stamps (e.g. $\sigma_0, \sigma_1, \ldots, \sigma_i$)

As an example of a timed language, the DOO pacemaker state diagram in Figure 3.9b is a simple timed automaton accepting the language

$$L = \{(((VP)(AP))^\omega, t) \mid \forall j.((t_{2j} = t_{2j+1} + 250) \wedge (t_{2j+1} = t_{3j} + 750))\}.$$

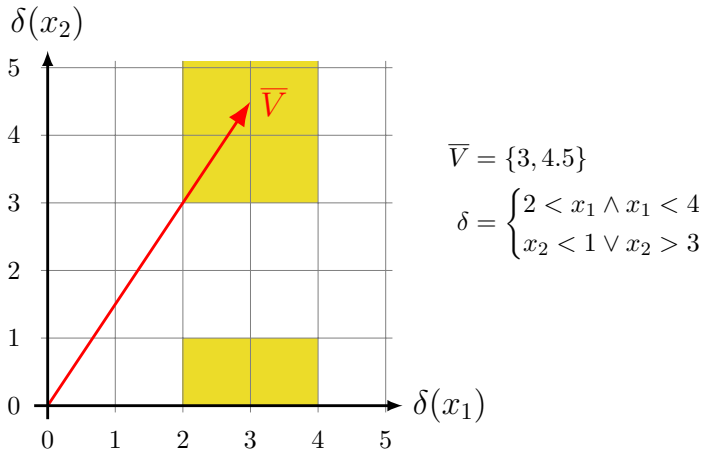


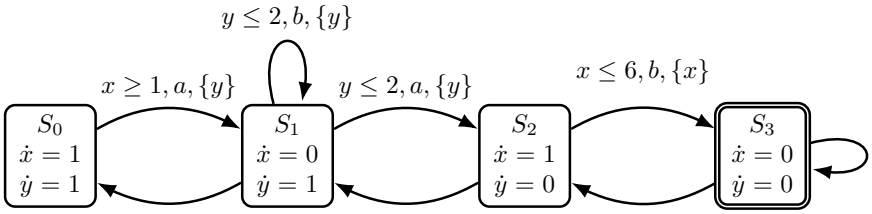
Figure 4.1: Evaluation of Clock Zones

A timed automaton extends finite automaton by adding a set of clocks X. A **clock valuation** v is a function that assigns a real number value to each clock, $v : x \to R_{\geq 0}, \forall x \in X$ and $V = R_{\geq 0}^{|X|}$ is the collection of all valuations. A clock $x \in X$ can be reset to zero on any edge transition, thus for $Y \subseteq X$, $[Y \mapsto 0]v$ denotes the clock interpretations where each $x \in Y$ is reset to 0. To accept timed languages, boolean **clock constraints** $\delta \in \Phi(X)$ are assigned to the automaton edges where δ is recursively defined as:

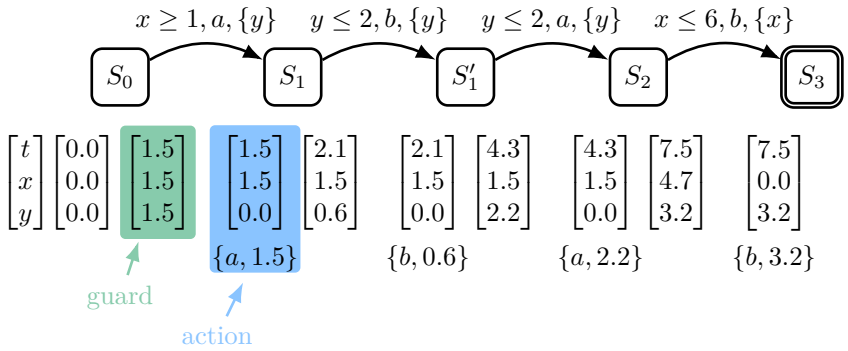
$$\delta := x \bowtie c \mid \neg\delta \mid \delta_1 \vee \delta_2 \mid \delta_1 \wedge \delta_2 \text{ where } \bowtie \in \{>, =, <\}.$$

Clock constraints form $|X|$ dimensional zones $\mathcal{Z}$ of acceptance as shown in the simple two clock example of Figure 4.1. The current clock valuations are shown as vector $\overline{V}$. If $\forall\delta \in \Phi(X)$ evaluate to true for a valuation V for X, V satisfies $\Phi(X)$.

Cassez and Larsen [28] extended the timed automaton, adding flexibility by allowing variable clock increment rates and the ability to pause a clock without reset via the **rate function** creating the stopwatch automaton, SWA. The rate function allows for measuring the time difference between two symbols of the automaton's alphabet and is denoted as the first derivative of the clock variable (i.e. for x, the rate function is $\dot{x}$ where $\dot{x} \in N_0$). For the remainder of this work, the rate function will be limited to $\dot{x} \in [0, 1]$ where the clock is either paused ($\dot{x} = 0$) or monotonically incrementing by 1 ($\dot{x} = 1$). Time is not required to advance and can stutter at a time point (e.g. $\forall x \in X, \dot{x} = 0$).



(a) Example Probabilistic Stopwatch Automaton



(b) A run of example Probabilistic Stopwatch Automaton

Figure 4.2: Probabilistic Stopwatch Automaton

A run over an SWA is an alternating sequence of time passage and actions of the form:

$$s_0 \xrightarrow{\tau_1} s_0' \xrightarrow{a_1} s_1 \xrightarrow{\tau_2} s_1' \xrightarrow{a_2} s_2 \ldots \xrightarrow{a_n} s_n.$$

where τ_i represents the passage of time before an action a_i occurs and a run of a SWT can be written as:

$$(a_0, t_0), (a1, t_1), \ldots, (a_i, t_i) \text{ where } t_i \leqslant t_{i+1} \text{ for all } i \geqslant 1, i \in R_{\geq 0}.$$

Figure 4.2 shows an example of a short Probabilistic Stopwatch automaton (PSA) accepting the timed language $(a, 1.5), (b, 0.6), (a, 2.2), (b, 3.2)$. The transition between each location in the figure is a two step process. First, the guard represents the state of the clocks on exit of the current location. Second, the action represents the state of the clocks on entering the next location along with the timed word created on the transition. The amount of time passed before checking the guard is nondeterministic.

Given a location $q \in Q$ and a valuation $v \in V$, a configuration of a PSA is the pair (q, v). In

configuration (q, v), action $a \in \Sigma$ become active when a span of time elapses such that the action guard is satisfied $t \in E(q, a)$. The time span and action are selected nondeterministically provided the action guard is satisfied and the action has been enabled. The next state after selection of an action is defined by $\delta(q, a)(Y, q')$ where $Y \subseteq X$ is the set of clocks to be reset.

Definition-1: Probabilistic Stopwatch Automaton

A probabilistic stopwatch automaton (PSA) is a tuple $\mathcal{T} = (Q, q_0, \Sigma, X, \gamma, E, \delta, F)$ where

- Q is the finite set of **locations**,
- $q_0 \in Q$ is the initial **location**,
- Σ is a finite alphabet of *actions*,
- X is the finite set of (clock or stopwatch) **variables**,
- $\gamma : Q \times \Sigma \to (X \to \{0, 1\})$ is the **rate** function,
- $E : Q \times \Sigma \to \mathcal{Z}$ is the **action guard**,
- $\delta : Q \times \Sigma \to \mathcal{D}(2^X \times Q)$ is the **transition function**, where $\mathcal{D}$ is a discrete probability distribution, and
- $F \subseteq Q$ is the set of final locations.

The following are some important subclasses of PSA:

(1) A PSA is a **probabilistic timed automaton** (PTA) if for all $q \in Q$, $a \in \Sigma$, and $x \in X$ that $\gamma(q, a)(x) = 1$ (e.g. clocks are always running). PTAs omit the description of the now static rate function and represent the PTA as the simplified tuple $(Q, I, \Sigma, X, E, \delta, F)$ and refer to its variables as *clocks*.

(2) A PSA is a discrete **stopwatch automaton** (SA) if for all $q \in Q$, $a \in \Sigma$, and $\varphi \in \mathcal{Z}$, we have that $\delta(q, a)$ if $P(q, a) = 1$ for some $a \in A$.

(3) A PSA is a **timed automaton** (TA) if it satisfies both of the previous conditions, i.e. all of the variables are clocks and transitions are non-probabilistic.

SWAs in general provide a simple visual method for introducing the logic of Duration Calculus

later in Section 4.1.4. Unfortunately, reachability for stopwatch automata is nondeterministic. A restricted SWA form that enables determinism will be presented in Section 6.2.

Stopwatch Automaton to MDP. With some manipulation of the PSA semantics, it can be shown that a PSA can be represented as an MDP with uncountable states and actions [43]. For a PSA $\mathcal{T} = (Q, q_o, \Sigma, X, \gamma, E, \delta, F)$ the equivalent MDP is $[\![\mathcal{T}]\!] = (S, s_0, A, T)$ where $S \subseteq Q \times V$ is the set of states; the initial state is $s_0 \subseteq Q_0 \times V$; the set of timed actions $A = R_{\geq 0} \times \Sigma$; and transitions $T : S \times A \to \mathcal{D}(S)$ such that for $(q, v) \in S$ and $(t, a) \in A$, $T((q, v), (t, a)) = d$ if and only if $\nu \oplus_{\gamma(q,a)} t \in E(q, a)$ and

$$d((q', \nu'))= \sum_{\substack{X \subseteq X \\ (\nu \oplus_{\gamma(q,a)} t)[X:=0]=\nu'}} \delta(q, a)(X, q').$$

for all $(q', \nu') \in S$. This is a handy realization that will enable modeling of complex real-time systems later in this document.

4.1.3 Linear Temporal Logic

Linear Temporal Logic (LTL) [116] is a formal language for describing the evolution of a timed system. An LTL formula φ is composed of a set of atomic predicates AP concatenated with logic symbols. It extends first order logic through the addition of temporal operators enabling formula to describe expected AP valuations in the future. A trace π of φ then is a sequence of AP valuations representing the system condition linearly over time increments $\pi = \sigma_0, \sigma 1, \ldots, \sigma_i$ where $\sigma_i \in AP$.

The syntax of LTL is defined as

$$\varphi_1, \varphi_2 ::= \bot \mid \top \mid p \in AP \mid \neg p \mid \varphi_1 \vee \varphi_2$$

where the conjunction operator $\wedge$ can be derived using DeMorgan's law.

The temporal operators are until U where φ_1 must hold until φ_2 holds and next X where φ_1 must hold in the next time increment σ_{i+1}.

$$\varphi_1, \varphi_2 ::= \varphi_1 U \varphi_2 \mid X \varphi_1$$

The LTL temporal modality operators are future F where φ holds sometime in the future and globally G where φ must always hold. These operators are syntactic sugar derived from U as $F ::= \top U \varphi$ and $G ::= \bot U \varphi$[1] .

Given a pointed word $(\pi, i) = \sigma_i$, a formula φ is **satisfied** when (π, i) fulfills φ at position i and is written $(\pi, i) \models \varphi$. If $\forall i, (\pi, i) \models \varphi,\ \pi \models \varphi$ it is said that π globally satisfies φ. Satisfaction over LTL can be shown inductively:

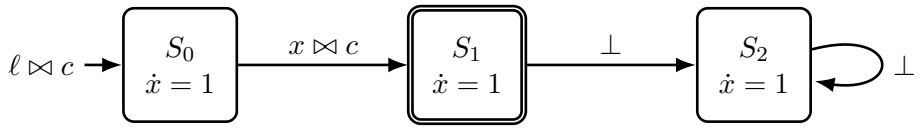
$$
\begin{aligned}
&(\pi, i) \models \top \\
&(\pi, i) \models p \qquad \text{iff} \quad p \in \sigma_i \\
&(\pi, i) \models \neg\varphi \qquad \text{iff} \quad \sigma_i \not\models \varphi \\
&(\pi, i) \models \varphi \vee \psi \quad \text{iff} \quad (\pi, i) \models \varphi \vee (\pi, i) \models \psi \\
&(\pi, i) \models \varphi \wedge \psi \quad \text{iff} \quad (\pi, i) \models \varphi \wedge (\pi, i) \models \psi \\
&(\pi, i) \models X\varphi \qquad \text{iff} \quad i + 1 < |\pi| \wedge (\pi, i + 1) \models \varphi \\
&(\pi, i) \models \varphi U \psi \quad \text{iff} \quad \exists k.(i \leq k < |\pi| \wedge (\pi, k) \models \psi) \wedge (\forall j.(i \leq j < k) \rightarrow (\pi, j) \models \varphi)
\end{aligned}
$$

4.1.4 Duration Calculus

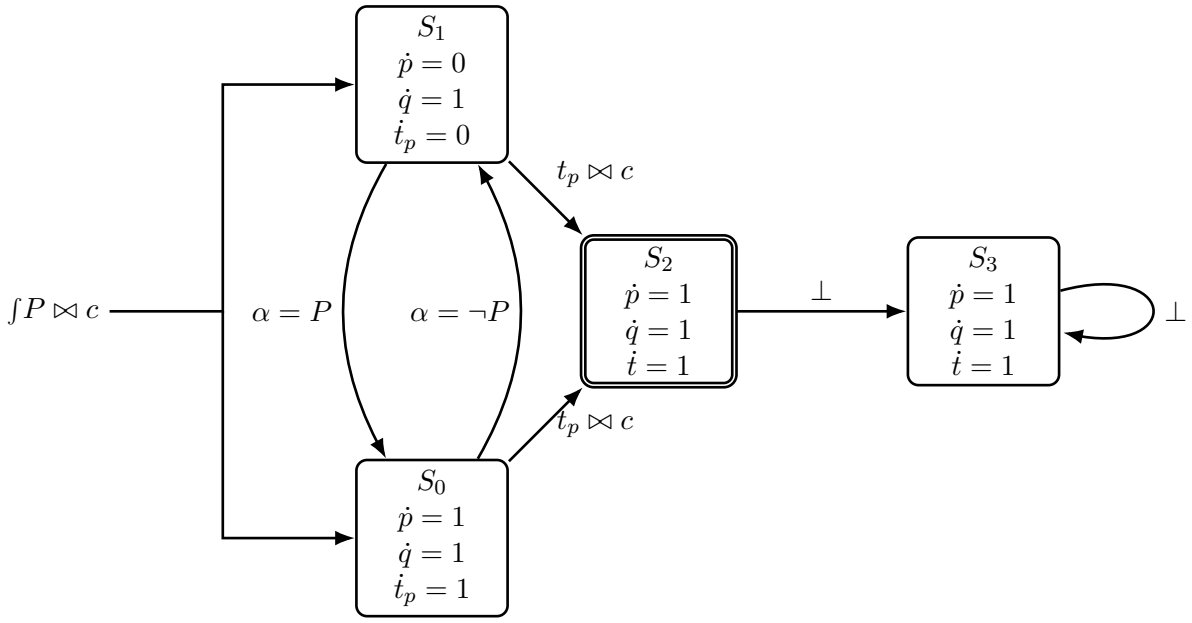
While MDPs provide an expressive formalism for modeling stochastic system processes, they provide no mechanism to represent temporal relationships between events. Likewise, formal logics such as LTL and Computational Tree Logic [36], CTL, allow for stating complex system requirements related to actions and events yet also lack a descriptive syntax for specifying passage of discrete time periods [82]. Logics such as Metric Temporal Logic [84], MTL, and Signal Temporal Logic [97], STL, provide mechanisms for describing real-time systems but lack the expressive richness for many real-time events such as counting occurrences of a periodic event within a time window or measuring the amount of time a signal holds within a window.

Duration Calculus (DC) [30] is a formal language that adds several operators for timing events and describing sequences of timed events. In addition to atomic propositions, DC adds the

[1] In this dissertation, Future may be represented as $\Diamond$ and Globally as $\Box$ (e.g. $\varphi ::= F\varphi \mid G\varphi$ or $\Diamond\varphi \mid \Box\varphi$)



(a) Probabilistic Stopwatch Automaton for measuring length of time



(b) Probabilistic Stopwatch Automaton for measuring time P holds over span c

Figure 4.3: Probabilistic Automaton for DC Measurement Operations

ability to explicitly state the duration D of a sequence of atomic propositions and measurement M of time spans or occurrences of an atomic proposition within a time span.

DC's measurement operators provide a means to describe lengths of time and how long or how often a proposition holds during a set duration. The length ℓ operator measures the passage of a length of time and allows comparison against a constant of the form $\ell \bowtie c$ where $\bowtie \in \{<, \leq, =, \geq, >\}$ and $c \in Q$. The length operator is diagramed as a stopwatch automaton in Figure 4.3a. Starting in State S_0 a timer $x \in X$ where X is the set of all clocks in the PSA and its rate function $\dot{x} = 1$ begins measuring an accumulation of time. When the comparison criteria, $x \bowtie c$, is met, S_0 transitions to the accepting state S_1.

Time in DC is dividable into a finite number of subintervals and atomic proposition (AP)

states hold constant almost everywhere over the interval. This allows for integrating over the subintervals of a period to determine the duration that P held within the period. This can be seen in Figure 4.3b where the automaton transitions between states S_0 and S_1 with the changes in the state of P. When $\neg P$ holds in state S_1, the rate function for timer x_P is set to zero to halt measuring passage of time.

The final measurement function is summation where the number of time points when P is asserted are counted. This count is then compared to a constant $\sum P \bowtie c$ where $c \in N$.

DC's duration syntax adds two new operators, duration $\lceil P \rceil$ and chop $\frown$. Given an interval of time d defined by two arbitrary end points b and e where $b \leq e$, the range operator $\lceil B \rceil^d$ is a valuation of P over an interval $\ell = d$ of time, $(V, [b, e])$ where $d = e - b$. This is demonstrated by the stopwatch automaton in Figure 4.4a. Given a clock $x_P \in X$ whose rate function $\gamma = 1$, the proposition P holds during state S_1 where clock x_P had a valuation of b when entering and e when exiting the state for a total duration of P holding in the state of $d = e - b$. The automaton accepts in state S_2 when $\ell = d$ time has passed with P holding in the previous state.

Figure 4.4b shows the special case where $e = b$. Denoted as $\lceil P \rceil^\bullet$ this operator specifies that P holds for a time interval of $\ell = 1$. In Figure 4.4b the acceptance criteria is a single transition from states S_0 to S_1 where P holds.

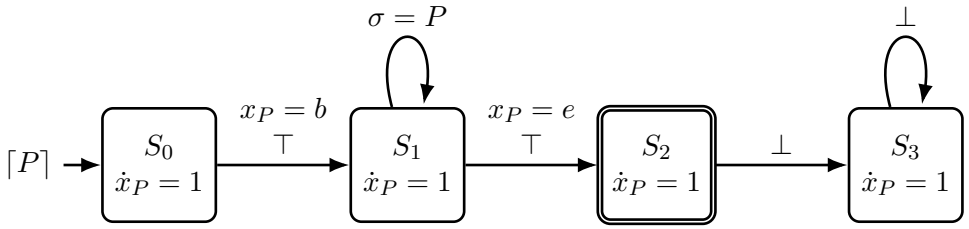
The DC chop operator $\frown$ allows creation of complex timed sequences consisting of a prefix and suffix where the prefix predicates must hold for a duration before the suffix holds. Thus, $\lceil P \rceil \frown \lceil Q \rceil$ states P must hold for a period of time and is immediately followed by Q holding. This is demonstrated in the stopwatch automaton of Figure 4.4c.

For some run $\pi = \lceil P \rceil \frown \lceil Q \rceil$ and the definitions of chop and measure given previously, we can formally state duration as:

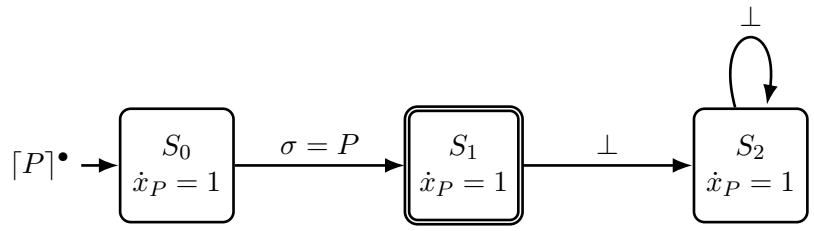
$$\lceil P \rceil^d = ((\ell = d \wedge \Box \lceil P \rceil) \frown \top)$$

$$\lceil \neg P \rceil^d = ((\ell = d \wedge \Box \lceil \neg P \rceil) \frown \top).$$

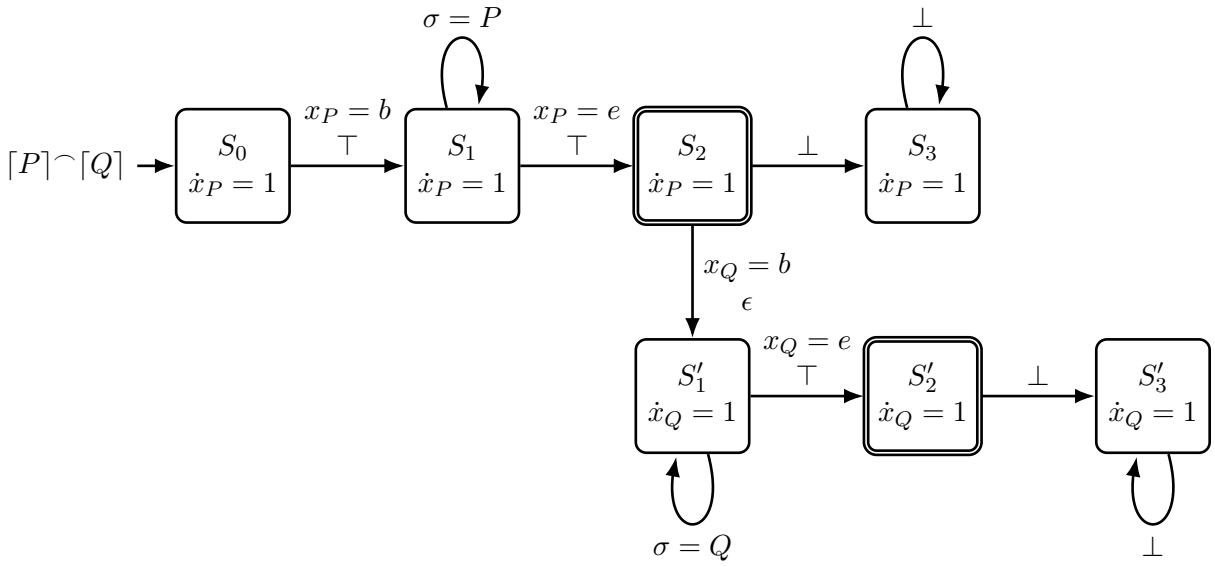
As an example of the expressive richness of DC, the property "**each of the propositions**



(a) Probabilistic Stopwatch Automaton for DC Range



(b) Probabilistic Stopwatch Automaton for DC Point



(c) Probabilistic Stopwatch Automaton for DC Chop

Figure 4.4: Probabilistic Automaton for DC Duration

p, q, r **are true exactly once**" can be expressed as:

$$\bigwedge_{x \in \{p,q,r\}} [(\top^\frown \lceil x \rceil^\bullet {}^\frown \top) \wedge \neg(\top^\frown \lceil \neg x \rceil^\frown \lceil x \rceil^\bullet {}^\frown \lceil \neg x \rceil^\frown \lceil x \rceil^\bullet {}^\frown \top)].$$

For the remainder of this paper, the notation $\lceil P \rceil$ will be used when the duration for which P holds is unspecified.

Definition-2: Duration Calculus Syntax

Given a set $P \in AP$ of finite set of atomic propositions, we define the syntax of DC formula as follows:

$$\begin{aligned} P &::= \bot \mid \top \mid x \in AP \mid P \vee P \mid \neg P \\ D &::= \lceil P \rceil \mid \lceil P \rceil^{\bullet} \mid D \vee D \mid \neg D \mid D^{\frown} D \mid M \\ M &::= \ell \bowtie c \mid \int P \bowtie c \mid \sum P \bowtie c \text{ where } \bowtie \in \{<, \leq, =, \geq, >\} \end{aligned}$$

DC formulas are evaluated over timed traces σ generated from runs of a probabilistic stopwatch automaton system model $\mathcal{T} = (Q, q_0, I, \Sigma, X, \gamma, E, \delta, F)$, where the language alphabet consists of the atomic propositions $\Sigma = 2^{AP}$ and a reference interval $I = [b, e]$ where $b \leq e$ and $b, e \in \mathbb{N}$ range over the indices $0, \ldots, n$ of the timed trace $\sigma = (s_0, \tau_0) \ldots (s_n, \tau_n)$. The satisfaction of a DC formula ψ evaluated on timed trace $\sigma = \langle (s_0, \tau_0), (s_1, \tau_1), \ldots \rangle$ with respect to an interval $[b, e]$ and is denoted as $(\sigma, [b, e]) \models \psi$.

For a timed trace $\sigma = \langle (s_0, \tau_0)(s_1, \tau_1) \ldots (s_n, \tau_n) \rangle$ and propositional formula P, we say $(\sigma, i) \models P$ iff $s_i \models P$. Given a DC formula ϕ, the DC satisfiability problem is to decide whether there exists a timed trace σ and interval I such that $(\sigma, I) \models \phi$.

The satisfaction of other DC formula is defined inductively:

(1) $(\sigma, [b, e]) \models \lceil P \rceil$ iff $b{<}e$, and $(\sigma, t) \models P$ for all $b{<}t{<}e$;

(2) $(\sigma, [b, e]) \models \lceil P \rceil^{\bullet}$ iff $b{=}e$ and $(\sigma, b) \models P$;

(3) $(\sigma, [b, e]) \models D_1 \wedge D_2$ iff $(\sigma, [b, e]) \models D_1$,$(\sigma, [b, e]) \models D_2$;

(4) $(\sigma, [b, e]) \models \neg D$ iff $(\sigma, [b, e]) \not\models D$;

(5) $(\sigma, [b, e]) \models D_1^{\frown} D_2$ iff there exists a point $b \leq z \leq e$ s.t. $(\sigma, [b, z]) \models D_1$ and $(\sigma, [z, e]) \models D_2$;

(6) $(\sigma, [b, e]) \models \ell \bowtie c$ iff $(\tau_e - \tau_b) \bowtie c$ holds;

(7) $(\sigma, [b, e]) \models \int P \bowtie c$ iff $\sum \{\tau_{i+1} - \tau_i \ : \ (\sigma, i) \models P\} \bowtie c$;

(8) $(\sigma, [b, e]) \models \sum P \bowtie c$ iff $|\{i \ : \ (\sigma, i) \models P\}| \bowtie c$.

Using the chop modality $\frown$, a bit of syntactic sugar allows the eventually modality $\Diamond D \stackrel{\text{def}}{=} \top \frown D \frown \top$ and globally modality $\Box D \stackrel{\text{def}}{=} \neg\Diamond\neg D$ to be derived. Additionally, the integral duration modality $\ell \in N$ denotes an integral interval $(\sigma, [b, e]) \models \ell \in N$ iff $\tau_e - \tau_b \in N$.

Duration Calculus Decidability. The unrestricted use of the measurement operators $\int P$ and $\ell \bowtie c$ over dense time (e.g. $t \in R_{\geq 0}$) leads DC to being undecidable [29]. Starting with the undecidable condition of a halting two-counter (Minsky) machine, Chaochen, etal [29] showed decidability can be restored through eliminating the $\int P$ operator and restricting to discrete time. With these restrictions, a decidable subset of DC with the time measurement variable ℓ and chop operator allows for expressing timed durations as shown for $\lceil P \rceil$ earlier.

Another option for recovering decidability is the removal of the real-time duration measurements $\ell \bowtie c$ and $\int P \bowtie c$ creating the logic subset Discrete Duration Calculus, DDC. The tool DCVALID [114] translates DDC to a deterministic finite automaton to check for satisfiability.

Neither of these options for recovering decidability are sufficient for learning a PTA due to their restrictions to discrete time. This will be addressed later in Chapter 6.2.

4.2 Machine Learning Algorithms

Machine learning (ML) is a branch of AI (Figure 4.5a) focused on learning tasks and data patterns through rewards. Modern ML solutions predominantly are composed of large networks of nodes that emulate the learning process of brain cells. They organize organically to form a generalized solution during the training process based on statistical patterns seen in the data used for training. Today, these patterns are largely indecipherable by human observers but they can be queried by providing fresh input data and testing the given result to what was expected.

For small ML networks with limited nodes and input data symbols, it can be possible to exhaustively test the final results. Large, deep networks are unable to directly represent all possible outcomes and must statistically interpolates between data points to generate solutions. It is also not feasible to train a deep network to all possible inputs when that data is infinitely large such as

the real numbers. This complicates verification and validation as exhaustive testing is not possible, leaving open the possibility of inaccurate performance when deployed for use. For safety critical systems, this would be unacceptable.

The validation and verification problem extends to how the network is deployed for use as well. Many times, it is desirable to continue training the network once deployed to strengthen what has already been learned and incorporate new information not seen in the original training data. By allowing continued training though without test, the network may begin to change such that it no longer maintains basic safety requirements and cause harm. Preventing training once deployed, however, limits one of the strengths of ML — to continually adapt to a changing environment.

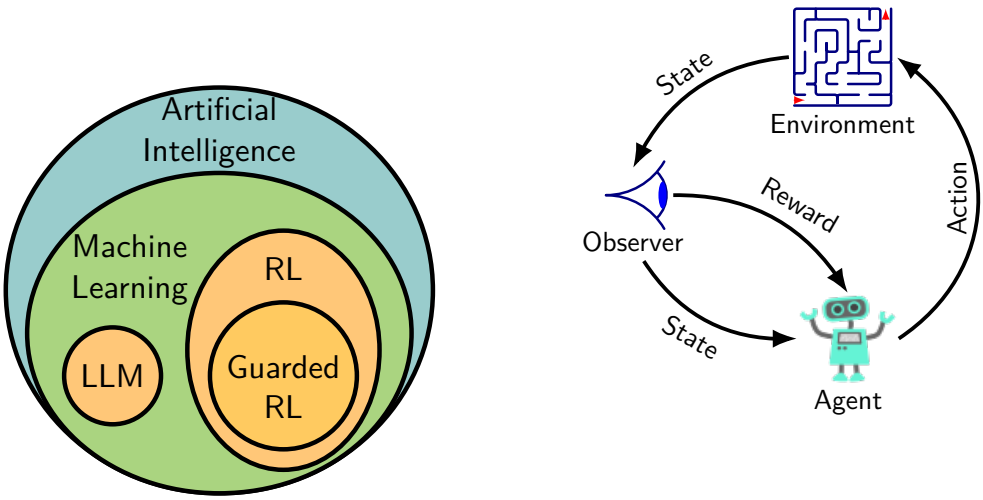
There are solutions that can enable the full power of ML for continued learning while remaining safe. This section introduces the basic technologies that will be used in later chapters where the safety versus learning conundrum will be addressed.

Figure 4.5a shows the areas of ML that will be described here and utilized in later chapters. Reinforcement Learning (RL) is a branch of ML for learning system control. Guarded RL adds an external observer to review the RL agent's decisions and overrides them when they would lead to undesired behavior. Large Language Models (LLMs) are another form of ML that specializes in recognizing patterns in data, statistically predicting the next value to occur in a given data input.

4.2.1 Reinforcement Learning

Reinforcement learning is a method to find the optimal policy for interaction with a dynamic, possibly partially hidden environment. The RL agent typically begins with no foreknowledge of the environment and uses trial-and-error to associate actions with environmental conditions based on a reward system that penalizes poor decisions. RL differs from other forms of ML such as supervised and unsupervised learning as it works in real-time with an evolving environment as opposed to fixed input data.

Figure 4.5b depicts the typical RL learning process. The RL agent, trying to find an optimal policy, begins by analyzing the current state of the environment and selects an action to take. The



(a) Partial AI Hierarchy (b) Reinforcement Learning Algorithm

Figure 4.5: AI overview

environment responds to the action and updates its state. An interpreter, sometimes referred to as a rewards machine, grades the agent's action based on whether the environment is closer or further from the desired condition. This grade is returned to the agent as a numerical reward value along with the new environment state. The RL agent continues to repeat this cycle, trying to reach the highest possible reward indicating an optimal policy. RL agents can be model-based, model-free or a combination of both depending on how the agent determines the next action.

Model-based RL. A model-based agent constructs a model of the environment based on next the state seen after taking an action. To select the next action, the agent performs a planning stage, cycling through the updated model state/action pairs to find the optimal path from the current state. This walk through the model to select the next action allows the agent to review multiple possible policies, making better use of limited experience to arrive at an optimal policy quicker than model-free methodologies. Because model-based RL selects the next action after reviewing its current environment model, it may be referred to as indirect reinforcement learning.

Model-free RL. A model-free, or direct reinforcement learning agent maintains no knowledge of the environment, learning instead through trial-and-error exploration and exploitation. Such

an agent uses received rewards to directly update the value function and selects its next action solely based on the updated value function. A model-free agent is simpler to construct but takes significantly more environmental interaction to converge to the optimal policy.

Model-based learning can arrive at a policy quicker than a model-free agent if the environment is deterministic. Thus, a model-based agent is computationally more efficient during training because it converges with fewer environmental interactions. Once trained, the model must still be traversed for each subsequent action at a cost in computational time. The model-free learning algorithm is significantly less complex allowing each new state and reward to be processed quickly. In a computationally constrained environment like an implanted, battery operated medical device, model-free learning is advantageous as the execution time to process the next environmental state is significantly shorter. For this reason, all agents used in the following chapters will be model-free.

Policy Optimization. For a finite MDP representing a stochastic real-time system, one problem is finding an optimal strategy for determining reachability. This can be approximated to computing an optimal strategy for a future discounted reward, a task RL is suited for. By setting the reward given for each action-state pair to the set $r \in \{0, 1\},\ r \in N_0$ where a reward of 1 is only given when reaching a target state with 0 for all other states, the average reward becomes equal to the reachability probability of the MDP.

Section 4.1.1 introduced MDPs and their value and action-value functions for measuring policy optimality (Equations 4.1 and 4.2 respectively). In model-free learning, the RL agent has no knowledge of the MDP's transition probabilities and must approximate optimality of the value function through backward induction using the Bellman equation [138]:

$$V_\pi(s) = \max_{a \in A(s)} \sum_{s' \in S} p(s', r|s, a) \left[r + \gamma V(s') \right]. \tag{4.3}$$

The value function for state s tests each possible action at s for the one that returns the highest future value. The value of each action is the sum of the probabilities of each potential next state $p(s', r|s, a)$ times sum of the reward that would be received for that state and its value. The environment MDP's transition table provides each next state's probability $p()$. The value function

is iterative as it is seeing the accumulated value of all possible future states embodied in $V(s')$.

Similarly, the State-Action or Q value searches for the best next action based on the value of the next states the action can reach:

$$Q_\pi(s,a) = \sum_{s' \in S} p(s', r|s, a) \left[r + \gamma \max_{a \in A(s)} Q(s', a) \right]. \tag{4.4}$$

Implementation of an RL agent can take many forms where each is designed to address unique goals or to trade off different algorithm strengths. Two of those, Q Learning and Policy Gradient networks, are introduced here as they will be used in later experiments.

4.2.1.1 Q Learning

In Q Learning [155], the RL agent maintains a table of the current valuation of each state-action combinations ($Q(s,a)$). The agent uses this Q-Table to walk through the environment from state to state by selecting the highest valued action for each state until reaching a goal state or a failure state. On each step, the agent updates the action values for the current state based on the reward received from taking the selected action. The optimal path from the current state s to the goal state is the optimal policy $V_\pi(s)$ and is defined as the path that returns the highest value.

Selecting actions from the Q-Table is termed **exploitation** as the agent exploits what it has learned so far. To prevent the agent from locking into a potentially less desirable policy of actions, an ϵ-greedy policy can be implemented where the agent will randomly select an action instead of the currently best action of the Q-Table based on a given probability ϵ to encourage **exploration** of alternate paths.

With model-less RL, the transition probabilities table of the environment model is unknown and thus the value and state-action functions cannot be directly calculated. Watkins [155] introduced an off-policy algorithm to approximate the state-action function Q_π:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a \in A(S)} Q(S_{t+1}, a) - Q(S_t, A_t) \right]. \tag{4.5}$$

One final parameter for Q-Learning is the rate of learning α. This parameter defines how strongly new information changes the Q-Table values. Changing the learning rate from a fractional

Algorithm 1 Q Table RL Agent [138]

Input: algorithm parameters: learning rate α and discount factor $\lambda \in (0, 1)$.
Output: Optimal Q values for each state action pair.
1: Initialize $Q((s, q), (t, a)) = 0$ for all $s \in S$, $q \in Q$ and $t \in N_k$ and $a \in A$.
2: **for** each episode **do**
3: $S := ((\ell_0, \mathbf{0}), q_0)$
4: **for** each step of episode **do**
5: Choose $A = (t, a)$ from Q-table using ε-greedy policy
6: Take action A observe R, S'.
7: $Q(S, A) = (1 - \alpha)Q(S, A) + \alpha[R + \gamma \max_{A'} Q(S', A') - Q(S, A)]$
8: Set $S' = S$.
9: **end for**
10: **end for**

addition to the updated Q value to a ratio between the present Q value and the maximum next reward gives the Q function:

$$Q(S_t, A_t) = (1 - \alpha)Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a \in A(S)} Q(S_{t+1}, a) - Q(S_t, A_t) \right]. \tag{4.6}$$

To visualize the learning process assume each row of the agent's Q-Table is a state of the MDP and each column is an action. Using Figure 4.5b and Algorithm 1 as a guide, for each step, the agent begins by indexing its Q Table by the current state and selects the action with the highest value (Line 5) and provides this action to the environment (Line 6). The environment updates its state based on the new action from the agent and passes the new state to the interpreter. The interpreter reviews the new environment state and if it's a goal state, sets the agent reward to 1, else 0. The reward and new state are given to the agent as an observation (Line 6). The agent updates the Q-Table for the old state based on the best reward that can be obtained from the new state (Line 7) using Equation 4.6. The agent then updates the current state value (Line 8) and the process repeats until the maximum allowed steps have been taken without solution, the agent enters a goal state, or enters a trap state.

Deep Q-Learning When the number of states and actions makes a Q-Table infeasible due to memory constraints to hold the table, Q-Learning can be replicated with a deep neural network. A deep-Q network (DQN) abstracts the Q-Table within the network such that a state-action pair is represented by the combination of weights in the network.

Algorithm 2 Deep Q RL Agent

Input: algorithm parameters: step size α, discount factor $\lambda \in (0, 1)$, Training batch size b, Log Length L.

Output: Optimal Q values for each state action pair.

Initialize network node weights $W_{i,j}$ to random values.
for each episode **do**
    $S := ((\ell_0, \mathbf{0}), q_0)$
    for each step of episode **do**
        Choose $A = (t, a)$ from Q-table using ε-greedy policy
        Take action A observe R, S'.
        Append S, S', R, A to log
        if $|log| = Len$ **then**
            $Batch$ = randomly pick b steps from log
            $y\,[b] = \max Predict(Batch)$
            $x\,[b] = Predict(Batch)$
            $Loss = MSE(x, y)$
            Update$(W_{i,j}, Loss)$
        end if
        Set $S' = S$.
    end for
end for

Being a network of interconnected nodes where weight changes are proportioned over each layer of the network, training after each action selection can cause the neural network to correlate actions to subsequent actions and retard training. To overcome this, experience or replay memory is implemented where the network is run for some number of steps without training while the step data (start state, next state, action, reward) are placed in a log of some defined length. When the log is filled, a batch of steps is randomly sampled, with or without replacement and this random batch is used for training the network to avoid correlation.

The final change to the Q-Learning algorithm deals with how to update the network after each action. Because the Q-Table is now an abstraction of the network, Equation 4.6 is no longer applicable. ML networks are updated by calculating the training error as a loss function comparing what was expected with what the network returned. This error is then back propagated through the network to update each node's weight contribution to the answer given.

A viable loss function can be created by returning to Equation 4.4. Assuming the probability

of the action taken to be 1, the equation simplifies to:

$$Q_\pi(s,a) = r + \gamma \max_{a \in A(s)} Q(s',a).$$

Using the left side of the equation as the desired output and the right half as the network's answer, the MSE loss function becomes:

$$cost = \left[Q_\pi(s,a) - (r + \gamma \max_{a \in A(s)} Q(s',a)) \right]^2.$$

The Deep-Q Network algorithm is shown in Algorithm 2. Similar to the Q-Table algorithm, Lines 5–6 take an action and collect the reward and next state. Line 7 places this information is placed in the log. When the log is full, Line 9 extracts a random subset as a batch, finds the maximum rewardful next state in Line 10 and has the network give its prediction in Line 11. The loss is calculated in Line 12 and the weights updated in Line 13.

4.2.1.2 Policy Gradient

Q and Deep Q Networks explicitly compute the expected rewards or $Q-$function by enumerating the state-action pairs to approximate an optimal policy where the $Q \times A$ state space may be quite large. Function approximators build upon the **policy gradient theorem** [139] to directly optimize the parameterized controllers [127, 64]. The results of the theorem allows for the computation of an approximate error gradient over the parameter space of the neural network without Q Learning's state enumeration of the unknown MDP. Doing so requires a method to provide rewards while learning that simulates the MDP.

Given a set of distributions $D(A)$ over a set of actions A, and an MDP $\mathcal{M}$ as a tuple (S, A, T), a **run** of $\mathcal{M}$ is an infinite sequence $(s_0, a_1, s_1, \ldots)$ such that $p(s_{i+1}|s_i, a_{i+1})>0$ for all $i \geq 0$. Let $Runs^{\mathcal{M}}(FRuns^{\mathcal{M}})$ be the set of runs (finite runs) of $\mathcal{M}$. A **reward machine** is a function $r : Runs \to \mathbb{R}$, implemented as a finite state machine [72], that provides a scalar reward for every finite run of $\mathcal{M}$.

Let $\pi_\theta : S \to D(A)$ denote a stationary strategy for $\mathcal{M}$ that is parameterized by π. We define our reinforcement learning problem as solving an optimization problem that maximizes the expected

long-term reward received by the strategy π_θ on $\mathcal{M}$. We are interested in maximizing the finite horizon ($T < \infty$) objective $J(\theta) = \mathbb{E}_{\pi_\theta}[G_{0,T}]$ where the discounted future rewards $G_{0,T} = \sum_{t=0}^{T} \lambda^t r_t$ and $\lambda \in [0, 1)$ is some discount factor and r_t is the reward obtained at time step t by taking some action a_t from the strategy π_θ at state s_t.

Policy Gradient seeks to refine a policy given a set of experience acquired from a given policy. The goal is to find the optimal policy by adjusting θ to maximize the objective function $J(\theta)$ using gradient ascent for a gradient step size α:

$$\theta_{new} \longleftarrow \theta + \alpha \nabla_\theta J(\theta). \tag{4.7}$$

The gradient of the objective function is the expected return of the discounted rewards G_0 starting from the initial state s_0 over a trace τ of length T per policy π_θ:

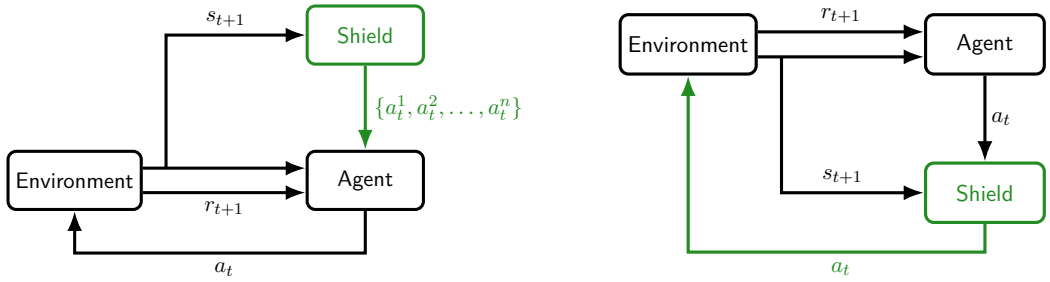
$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \left[G_{0,T} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right]. \tag{4.8}$$

4.2.1.3 Shielded Reinforcement Agents

With a software controller written in a computer language, one could test each decision in the code assuring that the correct action is taken when inputs are within and exceeding normal operating parameters to assure all requirements are met. One can assume that any combination of inputs that does not require a decision will meet requirements based on sampling testing and 100 percent testing of all possible input combinations is not required.

Conversely, RL attempts to approximate an optimal strategy that satisfies the specification. As an approximation it can be difficult or impossible to prove that strategy is completely correct for all input combinations. Without such assurance, it would be foolhardy to deploy an RL agent to operate a safety-critical system without concern for potential harm.

Shielded RL has been proposed [6, 20] as a method to regain safety assurance. The basic premise of shielded RL is the addition of a new component to the basic RL construct as seen in Figure 4.6 labeled as **Shield** between the learning agent and the environment. The shield represents



(a) Preemptive Shielding

(b) Post Posed Shielding

Figure 4.6: Shielded RL Methods[6]

the safety specification that the RL agent must satisfy and prevents any harmful or incorrect action selected by the agent from being taken.

A shield can be modeled as a two player safety game between the agent and the environment with the shield assuring the agent wins. Creating a shielded learning environment begins with a specification φ^S expressed in a formal language such as LTL or DC. The specification is translated to a safety automaton [85] $\varphi^S = (Q, q_0, \Sigma_S, \delta, F)$ where $F \in Q$ is the set of all safe states. The alphabet of the specification $\Sigma_S = \Sigma_I \times \Sigma_O$ is composed of the input Σ_I (controller actions) and output Σ_O (environment states) symbols.

The environment E to be controlled begins as a rewardful MDP $\mathcal{M}^E = (S, s_0, A, P, R, \gamma)$. To complete the safety game construction, the environment MDP is converted to an automaton $\varphi^{\mathcal{M}} = (Q_{\mathcal{M}}, q_{0,\mathcal{M}}, A \times L, \delta_{\mathcal{M}}, F_{\mathcal{M}})$ where the MDP states are translated to a set of labels describing the environment's current condition using an observer function $f : S \to L$. In this definition, $F_{\mathcal{M}}$ represents those states where the environment abstraction deviates from the actual environment. A step of $\varphi^{\mathcal{M}}$ is $q_{i+1} = \delta_{\mathcal{M}}(q_i, (l_i, a_i))$ for a run $q_0, q_1, \ldots, q_n \in Q_{\mathcal{M}}$.

The safety game G then is the cross product of $\varphi^{\mathcal{M}}$ and φ^S with winning states $F^g = (F \times Q_{\mathcal{M}}) \cup (Q \times (Q_{\mathcal{M}} \setminus F_{\mathcal{M}}))$. The shield now uses G to verify each action taken by the agent will not cause the environment to deviate from this set of winning states. If the shield finds an action would do so, the shield replaces the action to remain in a winning state.

Agent rewards for each action can be performed in one of two ways as seen in Figure 4.6. With preemptive shielding (Figure a), the shield is placed between the environment and the agent. Based on the current state of the environment, the shield will compose a set of legal actions from which the agent selects its next action. The agent receives a reward for the action it selected. Since the set of actions given to the agent are the only legal actions, the agent can never take an incorrect action and the reward reinforces selection of the best action.

In post posed shielding (Figure b), the agent is free to pick any action followed by the shield reviewing the action and correcting it if the action is incorrect. The agent receives a reward based on the action given to the environment by the shield. Positive or negative rewards for post posed shielding have drastically different effects on the agent learning and safety assurance. Negative rewards will deter the agent from taking the incorrect action again when presented with the same environment conditions. This has the unfortunate side effect of not being inclusive of all possible environment conditions where the action would be incorrect and therefore would not assure safety in all possible scenarios. Giving a positive reward for an incorrect action that has been corrected by the shield causes the agent to learn to always take the incorrect action each time without the knowledge that the shield is actually correcting it. This maintains system safety but may invoke the corrective abilities of the shield more often.

4.2.2 Sequential Data Networks

A common design challenge is finding patterns in sequential data such as a time sequence of controller actions or in speech recognition where symbol order dependency is important. Recent advancements in parsing and translating large bodies of text by Large Language Models between languages have been powered by algorithms capable of recognizing associations between sequential data symbols. Long Short-Term Memory networks were the primary method for sequential data processing until the advent of transformers. This is an introduction to these two important designs.

4.2.2.1 Long Short-Term Memory

A recurrent neural network (RNN) is one of several generic network designs for recognizing patterns in sequential data. The unique feature of an RNN is the cascading of previous results forward with each new symbol in the input sequence such that previous symbols seen affect how the next symbol is recognized. Generic RNNs suffer from vanishing or exploding gradients due to compounding the multiplicative effect of multiple sigmoids in a row due to the cascading architecture. A Long Short-Term Memory (LSTM) [69] is one of several specialized forms of the RNN designed to overcome the gradient issues of generic RNNs. The primary addition to the LSTM is the addition of a forgettable memory.

An LSTM, as shown in Figure 4.7a, is composed of three primary structures:

- The input gate controlling what information is added to the memory cell
- The forget gate defining when the memory cell is cleared
- The output gate deciding what information is output by the node.

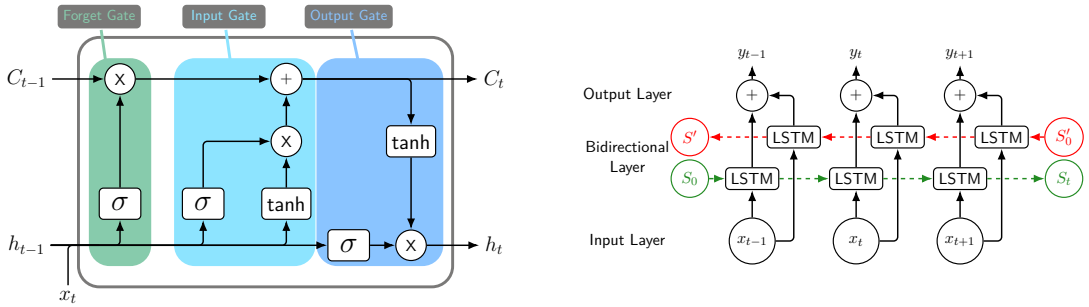
Starting from left to right in Figure 4.7a, the forget gate decides if older information should continue to be used or forgotten. The forget gate is simply a sigmoid function used to scale the cell's previous output based on its importance with regard to the sequence seen so far and the next input in the sequence. The function is applied to the combined next input vector x_t and hidden output vector from the node's previous output h_{t-1} and yields a values between 0 and 1. When the previous cell memory C_{t-1} is multiplied by this value, 0 will cause the memory to be cleared otherwise, the memory is diminished based on the magnitude of the value.

$$f_t = \sigma(W_{hf} h_{t-1} + W_{xf} x_t) \tag{4.9}$$

$$k_t = c_{t-1} \odot f_t \tag{4.10}$$

where W_{hf} and W_{xf} are the node weights.

Next is the input gate calculation. Like the forget gate, the input has an importance scaling function but with its own set of weights. It is used to scale an encoding of the previous hidden



(a) LSTM (b) Bi-LSTM

Figure 4.7: LSTM Network Architectures

state and the next input vector by way of a tanh function and this result j_t is added to the cell's previous value as scaled by the forget gate k_t to produce the cell's new output value c_t:

$$g_t = \texttt{tanh}(W_{hg}h_{t-1} + W_{xg}x_t) \tag{4.11}$$

$$i_t = \sigma(W_{hi}h_{t-1} + W_{xi}x_t) \tag{4.12}$$

$$j_t = i_t \odot g_t \tag{4.13}$$

$$c_t = j_t + k_t. \tag{4.14}$$

Finally, the output gate is another sigmoid scaling function that determines the magnitude of cell's output that should be passed back as the new hidden state h_t:

$$o_t = \sigma(W_{ho}h_{t-1} + W_{xo}x_t) \tag{4.15}$$

$$h_t = o_t \odot \texttt{tanh}(c_t). \tag{4.16}$$

Bidirectional LSTM. Bidirectional LSTM [61] or BiLSTM describes an RNN architecture using two LSTMs per node as seen in Figure 4.7b. An LSTM processes a sequence in the forward direction allowing it to predict upcoming symbols but suffers from losing context over long sequences. The BiLSTM simultaneously processes the input vector forwards and backwards enabling the node to understand the context of a new symbol both in the past and future.

The basic idea of the BiLSTM is the dual direction processing of the forward and backward data passes. In the forward data pass, the data is passed to the forward LSTMs one symbol at a

time, updating its output and memory on each symbol. Likewise and simultaneously, the backward data pass is sending the input vector in reverse order to the backward LSTMs. The outputs of the forward and backward LSTMs are combined to make the cell output.

4.2.2.2 Transformer Networks

In 2017 transformer networks [149] were proposed as an alternative method to analyze sequential data that does not rely on RNN based architectures. Transformers utilize an attention mechanism to overcome RNN restrictions on identifying dependencies in long input sequences.

The key feature of a transformer is the attention function. RNNs process an input sequence symbol by symbol to detect dependencies between symbols in the sequence. The attention function searches an entire sequence looking for symbols relevant to the task at hand such as finding nouns related to places. To begin, each x_i of a sequence $\{x_1, x_2, x_3 \dots x_t\}$ is a position-encoded word embedding vector of the original input symbols. During network training, query q, key k, and value v vectors are learned for each x_i through the creation of weights W_q, W_k, and W_v respectively.

Concretely, an attention function operates as follows. Given an input vector x_i, a query is created using its trained query weights. It is compared against the keys of the remaining vectors x_j creating an index vector of symbols with the highest similarity to the original. This index is then used to lookup the values for each symbol most similar to the query H_{ij}. Formally,

$$q_i = x_i \cdot W_q \tag{4.17}$$

$$k_j = x_j \cdot W_k \tag{4.18}$$

$$v_j = x_j \cdot W_v \tag{4.19}$$

$$h_{ij} = \texttt{softmax}\left(\frac{q_i \cdot k_j}{\sqrt{dim(d_k)}}\right) v_j \tag{4.20}$$

The value $\sqrt{dim(d_k)}$ scales the dot product $q_i \cdot k_j$ to improve convergence when $q_i \cdot k_j$ results in large values.

A transformer implements three attention functions:

- Self Attention — Attention between all symbol in the input sequence.

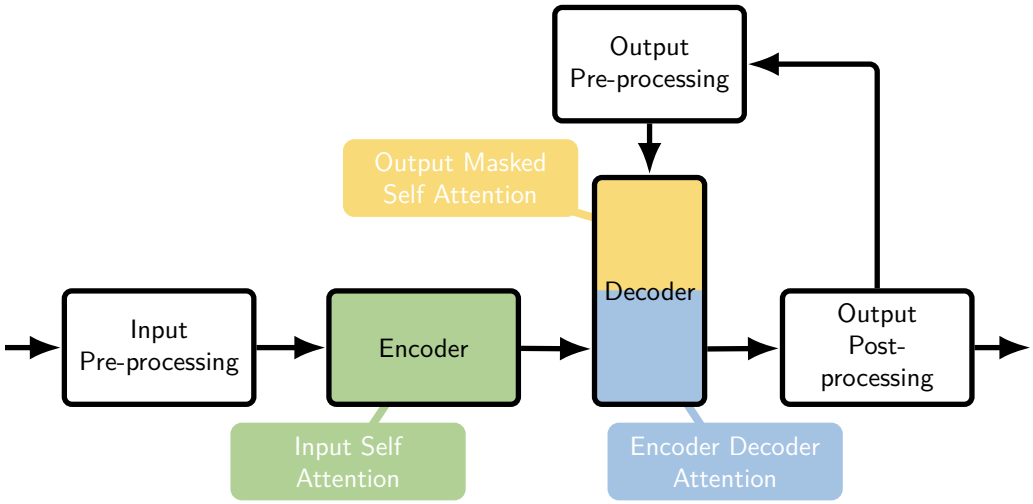


Figure 4.8: Transformer Attention Construction

- Encoder-Decoder Attention — Attention between the input and output sequences.
- Output Masked Self Attention — Attention between each output symbol and those preceding it.

The attention functions are arranged in the transformer as seen in Figure 4.8. Processing of an input sequence begins with input pre-processing where the symbols are converted to word embeddings. The embedded sequence then enters the encoder where input self attention is performed. The results of this passes to the decoder where it and the output from the previous step of the transformer are processed by the encode/decoder attention function. The output of the decoder is then post processed back to a symbol and this is the transformer's output. The output is also returned to the decoder for the next step of the sequence. Once trained, the transformer can return relational information about symbols.

4.2.2.3 Natural language processing

Natural language processing (NLP) is a field of computer science research focused on enabling computers to understand and communicate in human language typically combining the fields of machine learning and combinational linguistics. NLP research is composed of two primary research

areas. Natural Language Generation (NLG), the generation of linguistically correct text, and Natural Language Understanding (NLU), determination of meaning in text through semantic analysis. This section is not intended to give an in-depth theoretical treatise of NLP but provides an overview sufficient for recognizing the basic concepts and limitations. This discussion will lead later to a case study in Chapter 5 using NLP as part of the software development process.

In general terms, NLP encompasses a variety of algorithms for analysis of a string of symbols. Examples include word or sentence classification, text summarization, grammar correction, and language modeling. To accomplish this, symbols must be converted to a form usable for computer manipulation. In NLP this technique is known position-encoded word embedding, a process for converting symbols into unique multidimensional binary values. Global Positioning System (GPS) coordinates are a simple example vectorizing position into a unique three dimensional vector of latitude, longitude, and elevation for every point on earth.

Each dimension of a word vector represents some characteristic of the word such as unique ID values, parts of speech, or biologic phylum. Using these vectors, one can compare two symbols as to their uniqueness or similarities. For example, if one were to able to plot an n-dimensional vector for `cat` it would be seen close to `dog` and `pet` but far from `table`.

At its core, language modeling NLP is an algorithm that statistically predicts the next word or symbol in a series based on knowledge of previous symbols seen. Traditional NLP neural network configurations such as RNNs and CNNs have memory or ability to see multiple symbols simultaneously enabling tracking of the positional relationship and occurrence between symbols determining the next symbol. Transformers [58], described in Subsection 4.2.2.2, revolutionized this work enabling Large Language Models (LLMs) to work directly with the word vectors. Each transformer layer [149] appends additional vector information depicting newly identified relationships and context from within local and global symbol groupings. How word vectors evolve between transformer layers is not well understood though some progress has been made [71].

LLMs are trained over vastly larger corpora of text extracted from across the internet. As the corpus of text increases, language relationships strengthen and the ability to recognize the next

symbol based on internal analysis exceeds traditional methods for language modeling and translation as evidenced by current LLMs like ChatGPT-4 passing many standardized tests. Figure 4 of OpenAI's technical report on GPT-4 [1], the LLM behind the public ChatGPT-4 interface, shows the latest version scoring 80% or greater on 16 out of 26 standardized tests.

It is important to stress that an LLM is a NLP algorithm selecting the next symbol in a sequence. No LLM has yet shown contextual understanding of the text being generated and this leads to errors termed *hallucinations* or fictional generation [160]. In a hallucination text, the LLM creates factually incorrect or nonsensical responses. There have been multiple surveys [71, 160, 93] classifying hallucination types and recent work on detecting and mitigating [142] hallucinations. Hallucinations are an open field of research and any use of LLMs is constrained by this limitation.

4.3 The Software Development Process

In 1970, Royce [120] was the first to describe the development process for large software products. He observed the process was composed of incremental steps of refinement from specification down to testing an implementation. The process he described would become known as the waterfall method as depicted in Figure 4.9 due to the visualization of requirements flowing down to the next process step symbolized by green arrows in the figure. When errors were detected at a step in the method, they should ideally flow back to the previous step for correction, depicted in the figure by red arrows. Royce, however, recognized that, in many cases, an error may ripple backward multiple steps before a solution could be devised as the error may not be one of a coding mistake but of an incorrect assumption or model made earlier during specification or refinement. Critics of the waterfall model identify implications of distinct development order where one step should not begin until the previous completes and it would fall out of favor for many as emphasized by MIL-STD-498's discouragement in 1994 of its use.

In an attempt to address the short comings of the Waterfall model, development of the V Model, shown in Figure 4.10, began in the late 1980's. Forsberg [54] described the "Vee" model in 1991, derived from NASA's Software Management and Assurance Program (SMAP). Concurrently

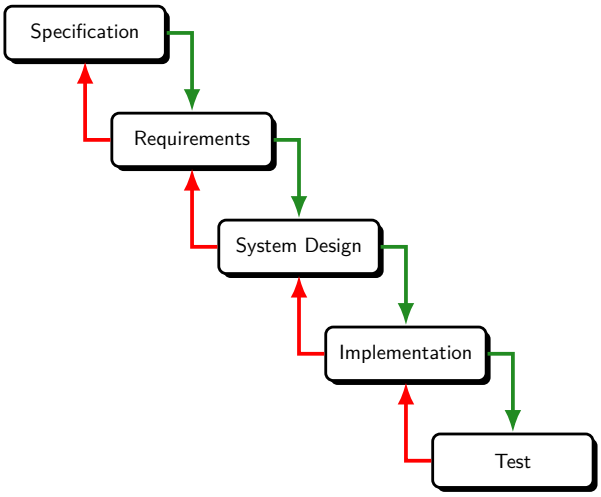


Figure 4.9: Waterfall Development Model

in Germany, the SEU-WS project would develop and publish the first version of the equivalent V-Modell also in 1991 [80] as the standard for software creation in the German Bundeswehr. The V-Modell continues to be maintained and updated by Weit e.V. where the current version is V-Modell XT. The V model is commonly illustrated in software development standards that require stringent documentation such as IEC-62304.

The V Model splits the development task into two parts, the design phase on the left and the verification and validation phase to the right. The horizontal rows represent the design hierarchy, matching the design depth on the left with the associated test phase on the right. The left half of the V works similarly to the Waterfall model as progress moves down the chart (green arrows) and errors (red errors) discovered move back up. Once implementation has completed, progress moves up the right side of the V through higher levels of testing until final product validation is completed, each level tested to the associated specification (blue arrows). Errors found during test reflect horizontally across the diagram (red arrows) to the associated design level signifying the most probable location where the incorrect assumption or modeling error occurred.

Over time, increasing software product complexity exposed weakness of the V Model [125]. Of note, the V Model is dependent on well known product definition as specification churn disrupts follow on phases. Like the criticisms of the simpler waterfall model, rework and iterative devel-

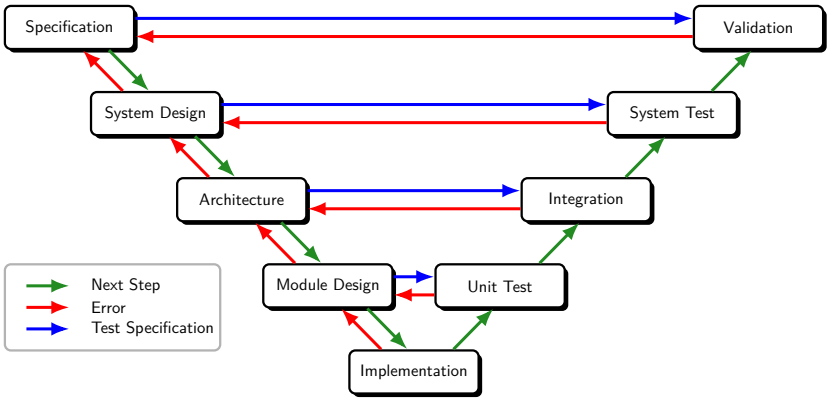


Figure 4.10: Software Development V Model

opment techniques are difficult to represent. Likewise, new programming paradigms such as Test Driven Development, Extreme Programming and Agile are incompatible. One of the strengths of the V model, strong documentation requirements at each level, is also its weakness as the burden of documentation updates can cause changes at lower levels to not be reflected at higher levels allowing the actual design to diverge from its documentation.

In 2001 the Object Management Group (OMG) began the Model-Driven Architecture (MDA) initiative [19] to define a development process compatible with modern design principles and project sizes. MDA may also be referred to as Model Based Design (MBD), Model Based Engineering (MBE), and Model Driven Engineering (MDE) [110] though each of these has some unique differences. Like the V model, each step of the MDA design process is rigorously documented though substituting computer readable models for the V model's extensive native language documentation at each step of the process. The key element of MDA is the Concept Graph (CG). A CG is a language used to express abstract or meta models of the system to be designed.

Model creation is the key to MDA with many important advantages over traditional NL documentation. First, MDA tools increase designer productivity by allow execution of the models, enabling early testing of behaviors and actions, theoretically allowing for fewer errors to escape into later phases and subsequent rework. This capability provides incentives to maintaining the models during the error correction process, addressing a major short coming of the standard V model in

reflecting all design changes throughout the documentation tree. MDA tools such as Rationale ROSE, Esterel Technologies SCADE Studio, I-Logix Rhapsody and Mathworks Simulink allow for generating the artifacts for the next state, be it a more refined model or code decreasing the probability of translation errors occurring between phase steps. Most importantly for this thesis, a CG based model can readily be translated to a formal language enabling the use of SAT solvers and other tools for testing satisfiability and correctness of the CG model. As a demonstrated example, Murugesan [113] focused on the FDA infusion pump initiative [51] with the goal of creating a model of an infusion pump that could be tested via formal methods and produced a correct by design software solution that could be validated in a physical infusion pump.

The CG is considered a Formal Specification to which all verification testing is performed. While this allows for checking that the implementation meets the original CG, it does not address validation to the original natural language specification. Chapter 5 concludes with a case study focused on translation of a natural language specification to a formal LTL specification.

Chapter 5

Formal Specifications From Natural Language Explanations

While the exploration and exploitation abilities of reinforcement learning can enable novel new therapies, it is only possible if the rewards machine used for training is correct by design. The concept of Correct by Design was introduced in Section 4.3 with the discussion of Model Driven Architecture and that section referred to some of the tools and procedures for translating formal logic specifications to automata and validating satisfaction of the specification.

Unfortunately, the vast majority of top level specifications are written in natural language and not a formal language, requiring an error prone translation before formal methods may be applied [156]. This has remained a serious impediment to Correct by Design. To understand what drives translation failures, Greenman, et al [62], performed an analysis of practitioner ability to translate from english to LTL, LTL to english, and LTL to traces. While their results showed errors among even seasoned practitioners for each task, the results found translation from english to LTL to be the most error prone of the three tasks. These results highlight the need for a better method for translating requirements to formal language.

There have been multiple efforts [65] to solve this translation dilemma with limited results. Recent advances in the AI field of Large Language Models (LLMs) for parsing and translating languages [1, 33, 143, 146] open new possibilities for accurate translation from natural language to formal language. As described earlier, LLMs do not understand the text given nor the text produced but generate a string of words most likely to occur in a sequence. This limits the LLM's ability to recognize the logic contained in a given string of text and its ability to construct valid

formal logic formula to represent it [107].

Because translation from natural to formal language continues to struggle for completeness and accuracy, automated extraction of LTL specifications from demonstrations is an active area of research [90, 2, 39]. In this method, the designer creates a model exhibiting the desired response to intended input or records an existing system's operation to be used as a demonstration database for specification extraction.

This chapter begins by looking at extraction of a specification from observation. Given a large dataset of surgical tool usage extracted from surgical video, an LLM will be tasked with extracting formal statements on tool usage. Realizing that the result may contain errors, this demonstration will be followed by a method to repair the results.

5.1 Deriving Specifications Using Large Language Models

A process is a series of steps and actions performed to achieve a goal. We are surrounded by process whether we're conscience of them or not. The morning routine of getting out of bed is a process of steps with the goal of getting that morning cup of coffee. A manufacturing line is an example of a process to build a product. Driving a car is an example of a process for operating a machine. Process analysis is a common system engineering task where the engineer observes and documents the process steps. The analysis may include any tools used and in what order, the skills or dexterity required by the operator, and how the product evolves over the course of the process. Once documented, a process can be analyzed for such items as inefficiencies, steps prone to error, and places where better tools, training or automation might provide improvement.

Surgical procedures are an invasive medical process intended to treat a patient malady. The process of many surgical procedures has traditionally been highly non-standardized. While various medical societies define best practices for specific procedures, practitioners, while conforming to the best practices, typically follow the more detailed process they were taught in medical school, during their residency, or based on local hospital tradition. Multiple studies [3, 12, 134] have shown that standardization of detailed surgical steps can lead to better patient outcomes and thus analyzing

surgical technique for conformance has become relevant to hospital managers.

Minimally invasive surgery (MIS) encompasses a class of surgical procedures where several small incisions are made, only sufficiently large to allow passage of surgical tools. In order to perform the procedure, a camera tool is used to provide clinician vision of the surgical site and tools within the surgical field. The video from these cameras is increasingly being recorded and stored for later analysis. The volume of videos being created and stored, however, has far outstripped the capacity for manual review and automation is now being applied. Automated video processing [95, 154] enables large scale process analysis of the surgeon's technique and adherence to standardized procedural steps [41, 22, 70]. The automation allows for detection of which tools are in the surgical field, how long they remain in the field, and what they are currently doing. In some cases, critical structures of the body can be identified and the relationship of the structure to any visible tools can be compared to expected surgical practice.

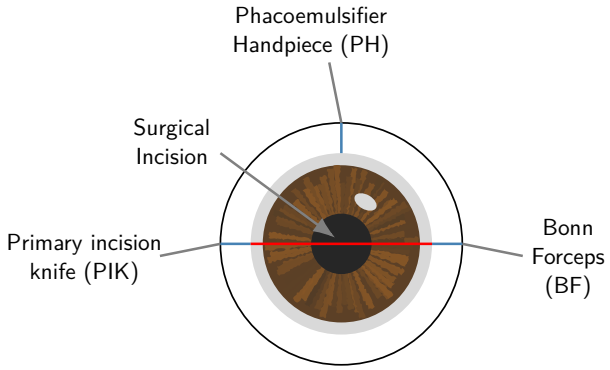


Figure 5.1: Typical surgical tool location during a step in cataract surgery.

AI analysis of surgical video is becoming a main stream technique throughout the medical and research fields. Seeking to extend video analysis to cataract surgery, the CATARACTS[1] challenge was created to encourage a competition between researchers to find the best methods for video analysis of phacoemulsification cataract surgery. This surgical procedure has five primary steps: (1) incision, (2) emulsification, (3) lens extraction, (4) new lens implantation, (5) closure. The goal of the analysis is to determine what tools are used in each surgical step and the order of

[1] `https://grand-challenge.org/All_Challenges`

usage where some tool usage may overlap each other and a tool may appear multiple times.

Like many surgeries, cataract surgery requires a precise sequence of tools, yet there is some variation based on individual surgeon's preference. The preferred sequence may have been written as a formal paper published in a societal journal or extracted from viewing a large body of surgical video. In either case, the result is a natural language specification and, again, the issue of translation into a form usable for grading clinicians or programming a surgical robot. An alternative option is to extract a formal language specification from the video analysis that codifies the tool sequence in an unambiguous specification.

Case Study Part 1

This chapter's case study is broken in two parts. First an LLM extracts a formal specification from video analysis followed by detection and correction of any errors in the LLM formula. In this first half of the case study, an LLM was given a natural language description of the surgical procedure, a series of short traces sampled from the CATARACTS database, and some examples of LTL formula. The LLM's task is to provide a candidate LTL formula describing tool usage that accepts the given traces. This case study used the ChatGPT 4.0 LLM. Two ophthalmologists provided this natural language description of the procedure as the textual input to the LLM:

> The Bonn forceps (BF) is employed to stabilize the eye. Side ports at 3 and 9 o'clock are made for paracentesis using a secondary incision knife (SIK). The anterior chamber is filled with viscoelastic using a viscoelastic cannula (VC), then Curvilinear Capsulorhexis is performed using Capsulorhexis cystitome (CC) by puncturing the capsule and lifting a small flap. The main port entry is made using a 2.8 or 3.2 Primary incision knife (PIK). The cataract is separated from the capsule by using Hydrodissection cannula (HC), while injecting fluid to loosen and free the lens. A viscoelastic solution is injected to fill the anterior chamber using the viscoelastic cannula (VC). Now the lens manipulator (MM) hook is passed through a side port to rotate the nucleus ensuring free movement of the nucleus. Next the anterior chamber is filled with viscoelastic solution through the viscoelastic cannula (VC). The phacoemulsifier hand-piece (PH) is then entered through the main port. During the phacoemulsification process, a phacoemulsifier hand-piece (PH) uses ultrasound to emulsify the nucleus of the cataractous lens. A lens manipulator (MM) may be introduced through a small side incision from 9 o'clock further to break down the hard cataract nucleus into smaller fragments. Once the emulsification of the lens nucleus is complete, an irrigation-aspiration (IAH) system is used to flush any remaining cortical material. Finally, acrylic intraocular lens is implanted using an implant injector (II) and precisely adjusted with a micro-manipulator (MM).

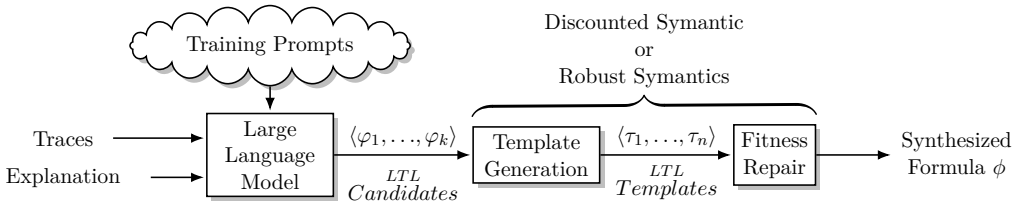


Figure 5.2: Method for synthesizing and correcting LTL formulas from traces and descriptions

Given the described input, the LLM produced the following formula:

$$\mathsf{G}((BF \vee SIK) \Rightarrow (\mathsf{X}(PIK \wedge \mathsf{X}(VC \wedge \mathsf{X}(PH \wedge \mathsf{X}(IAH \wedge \mathsf{X}(II \wedge \mathsf{X}(MM))))))). \tag{5.1}$$

When converted to a Büchi automaton, the resulting graph has 57 states, 48 accepting and 9 rejecting. The resulting formula describes many possible sequences of tool usage including a large number that are incorrect. It also loses most of the temporal relationships where tools may be used simultaneously or in a specific sequence. Additionally, the use of the implies allows for a procedure that never used the Bonn forceps — a key tool for holding the surgical site stable. More egregious is the placement of the globally (G) around the entire equation allowing for a cyclic aspect to the procedure that does not exist in real life. This last point appears to indicate that an LLM struggles with temporal properties not explicit in the syntax of the original input text.

5.2 Repairing LLM Derived Formulas

As the previous section showed, an LLM's ability to generate formal language formulas from descriptions and traces is not reliable. This section presents a process to correct the LLM's errors to produce a usable formula.

Beginning a correction process starts with assessing the given formula φ against a finite sampling of a trace demonstrating expected operation w. This is performed by a valuation function quantifying the fitness of φ in explaining w. Two valuation functions are introduced here based on recently introduced quantitative semantics: **Robust Semantics** [2] and **Discounted Semantics** [7] with the intent of observing differences between each. These valuation functions focus on two aspects of LTL structure, temporal and nesting simplicity.

The temporal formula Gp yields a simpler and more robust explanation of the trace fragment $\langle \{p\}\,\{p\}\,\{p\}\rangle$ than $p \wedge Xp \wedge XXp$. Simplicity is related to the levels on nesting in a tree structure representing the formula. With a sample $\langle \{p\}\rangle$, the tree structure of a formula p would have a single node whereas the tree structure of $p \wedge \neg q$ would have a more complex arrangement of a parent node and two leaves. The formula construction and correction process is driven by the temporal discounting parameter $0 \leq \alpha \leq 1$ and nesting discount factor $0 \leq \beta \leq 1$, which reward simpler formulas with fewer temporal and binary operators.

Robust Semantics. The robust semantics valuation function returns a positive value when a trace satisfies a formula and a negative value otherwise. The valuation magnitude indicates the level of satisfaction. Based on the robust semantics [2], given an LTL formula φ and a finite trace $w \in \Sigma^*$, the valuation for each LTL operator in $[\![\varphi, w]\!]$ is defined as follows.

$$[\![a, w]\!] = 2 \cdot [\![a \in w(0)]\!] - 1$$

$$[\![\neg a, w]\!] = -1 \cdot [\![a \notin w(0)]\!]$$

$$[\![\varphi \wedge \psi, w]\!] = \begin{cases} \beta \cdot [\![\varphi, w]\!] \cdot [\![\psi, w]\!] & \text{if } [\![\varphi, w]\!] \geq 0 \wedge [\![\psi, w]\!] \geq 0, \\ -1 & \textit{otherwise} \end{cases}$$

$$[\![\varphi \vee \psi, w]\!] = \begin{cases} \beta \cdot \text{avg}([\![\varphi, w]\!], [\![\psi, w]\!]) & \text{if } [\![\varphi, w]\!] \geq 0 \wedge [\![\psi, w]\!] \geq 0, \\ \beta \cdot \max([\![\varphi, w]\!], [\![\psi, w]\!]) & \text{otherwise} \end{cases}$$

$$[\![G\varphi, w]\!] = \beta \cdot \begin{cases} \sum_{i=0}^{|w|} \alpha^i [\![\varphi, w[i]]\!] & \text{if } [\![\neg\varphi, w[t]]\!] < 0,\ \forall t \leq |w|, \\ -1 & \text{otherwise.} \end{cases}$$

$$[\![F\varphi, w]\!] = \begin{cases} \beta \cdot \alpha^t [\![\varphi, w[t]]\!] & \text{where } t = \min\{j | [\![\varphi, w[j]]\!] \geq 0\}, \\ -1 & \text{if } [\![\varphi, w[t]]\!] = 0,\ \forall t \leq |w| \end{cases}$$

$$[\![(X\varphi, w)]\!] = \begin{cases} [\![\varphi, w[1]]\!] & \text{if } [\![\varphi, w[1]]\!] \geq 0 \wedge |w| > 1 \\ -1 & \text{otherwise.} \end{cases}$$

$$[\![\varphi U \psi, w]\!] = \begin{cases} \alpha^t \cdot [\![\psi, w[j]]\!] & t = \min\{j \mid [\![\psi, w[j]]\!] \geq 0\} \text{and } [\![\varphi,\ w[i]]\!] \geq 0, \quad \forall i < t, \\ -1 & \text{otherwise.} \end{cases}$$

Discounted Semantic. The discounted semantics valuation function returns a value in the range $[0, 1]$ where 0 indicates poor satisfaction. For a finite trace $w \in \Sigma^*$ and LTL formula φ, we define the valuation of φ wrt w, $[\![\varphi, w]\!]$ as follows:

$$\begin{aligned} [\![\top, w]\!] &= 1 \\ [\![a, w]\!] &= 1 \text{ if and only if } a \in w(0) \\ [\![\neg\varphi, w]\!] &= 1 - [\![\varphi, w]\!] \\ [\![\varphi \vee \psi, w]\!] &= \beta \cdot \max\{[\![\varphi, w]\!], [\![\psi, w]\!]\} \\ [\![\mathsf{X}\varphi, w]\!] &= \alpha [\![\varphi, w[1]]\!] \\ [\![\varphi U \psi, w]\!] &= \max_{0 \leq i < |w|} \left\{ \min \left\{ \alpha^i \cdot [\![\psi, w[i]]\!], \min_{0 \leq j < i} \left\{ \alpha^j \cdot [\![\varphi, w[i]]\!] \right\} \right\} \right\}. \end{aligned}$$

The two valuation functions described above follow Alur's work [7] with the addition of discounting for Boolean operators and restriction to finite traces.

The entire correction process roughly follows the robust semantics as defined by Afzal [2]. The first step is generation of candidate formulas. Following the robust semantic process, candidates were created by hand. As seen in the cataract case study, this manual task is replaced with an LLM. The LLM is requested to generate a number of candidate formula that could satisfy a given explanation of the desired formula and a sampling of positive traces. The valuation function then rates each LLM generated formula with the highest scoring moving on to the next step. This formula is parsed into a tree structure for the repair process. To convert the formula to a template, during the parsing process there is a small probability of substituting a blank node for the formula's actual operation or predicate. This step is followed by an algorithm that tries to best fit the given positive traces by substituting operations and predicates into the template's blank nodes.

The Janaka tool [63] embodies this process as shown in Figure 5.2 and detailed in Algorithm 3. Line 1 requests candidate formula from an external LLM. Line 2 scores each candidate

Algorithm 3 JANAKA Algorithm

Input: Traces T, Explanations E, Prompt P, Depth d, Formula return quantity κ
Output: Valuation
1: $\Psi \leftarrow$ API2LLM (T, E, P) ▷ API call to LLMs
2: **if** $\max_{\psi \in \Psi}\{[\![\psi, T]\!]\} \geq \kappa$ **then return** $\text{argmax}_{\psi \in \Psi}\{[\![\psi, T]\!]\}$
3: **else**
4: $\quad \Psi_{top} \leftarrow \text{top\_k}_{\psi \in \Psi}\{[\![\psi, T]\!]\}$ ▷ Returns top k fit formulas
5: $\quad \Psi_{temp} \leftarrow$ TEMPLATEGEN (Ψ_{top},d) ▷ Generate templates
$\quad$ **return** LTLREPAIR$(T, \Psi_{temp}, d, \kappa)$ ▷ Repair LTL from templates
6: **end if**

Algorithm 4 TEMPLATEGEN

Input: Set of LTL Formulas Ψ, Depth d
Output: Template Tree(Ψ)
1: Combine the given formulas Ψ to construct a parse tree Tree (Ψ)
2: **for** each visit to a node of Tree(Ψ) in breadth-first order **do**
3: $\quad$ **if** RANDOM$(0, 1) < 0.2$ **then**
4: $\quad\quad$ Replace the node in Tree(Ψ) by a template hole of depth $\leq d$
5: $\quad$ **end if**
6: **end for**
$\quad$ **return** Tree(Ψ)

Algorithm 5 LTLREPAIR

Input: Traces T, Templates Ψ, Depth d, κ
Output: Formula m
1: Construct tree constraint $\Phi_d^{template}$ from Ψ
2: Construct trace constraint Φ_d^T
3: $\texttt{fitness}, m \leftarrow \max \left(\sum_{\tau=1}^{|X|} y_{1,0}^{\tau} \mid \tau \in T \right)$ such that $m \models \Phi_d^{template} \wedge \Phi_d^T$
4: **if** $\texttt{fitness} \geq \kappa$ **then return** the formula constructed from m
5: **else return** False
6: **end if**

with the valuation functions. If a formula valuation exceeds a set threshold, the formula is returned and the process ends. Otherwise, the highest valuation formulas are collected in Line 4 and converted to templates in Line 5. With the templates, Line 6 attempts to repair the templates to satisfy the given traces. Algorithm 4 shows the template process of parsing and randomly blanking tree nodes. Algorithm 5 performs the repair operation.

Case Study Part 2

Returning to the cataract case study, the LLM formula was submitted to JANAKA for cor-

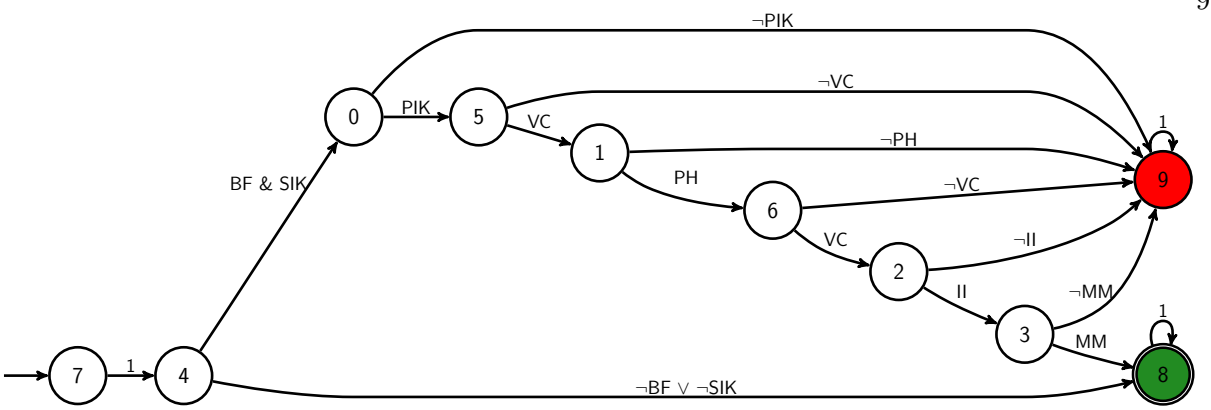


Figure 5.3: Automaton based on the repaired surgery formula

rection. The tool was able to recognize replacing G with X at the beginning of the formula would produce better fitness:

$$\mathsf{X}(((BF \wedge SIK) \Rightarrow \mathsf{X}((PIK \wedge \mathsf{X}((VC \wedge \mathsf{X}((PH \wedge \mathsf{X}((VC \wedge X((II \wedge \mathsf{X}(MM))))))))))). \quad (5.2)$$

The updated formula is shown as an automaton in Figure 5.3.

5.3 Conclusions

The advent of modern AI architectures has brought a wealth of new data sources by automating the laborious task of manual review and annotation.as evidenced by the rapid increase automated medical surgical video annotation. Recognizing the availability of this new data inspires the first question posed in this chapter of whether a formal language process specification can be derived from examples. Secondly, can the greatly increased language translation ability of modern Large Language Models could be used to do the extraction. Finally, acknowledging the fact that LLMs still generate erroneous results, is it possible to identify and correct the results.

The results presented were mixed but promising. In the cataract case study, the formula extracted by the LLM using a description of the procedure and positive examples resulted in a formula that exemplified a great deal of the logic embedded in the source material. It was, however, still incorrect. The result was also somewhat naive in capturing only a portion of a much wider and richer scope of possible surgical tool combinations. One can draw several conclusions from these

results. First, LLM's struggle with detection of concurrency. As seen in the original extracted Formula 5.1 the Bonn Forceps, required through out the procedure to hold the surgical site stable has been extracted as a single event. Additionally, the micro manipulator can optionally be used concurrently with the phacoemulsifier hand-piece. This concurrency is not recognized either.

The repair process was successful in returning a formula that better fit the original narrative. Using templates in the repair process has its limitations. Namely that it is only able to repair what already exists by swapping LTL operators. The process currently cannot reintroduce missing required predicates to the original formula.

Chapter 6

Formal Pacemaker Specifications

A specification is the essential method for communicating design objectives. It is a collection of statements defining essential performance including safety and reliability expectations. The specification is the starting point for any design, regardless of design methodology (see Section 4.3). Getting specification requirements correct for complex systems is a difficult task [88]. Requirements can conflict, be incomplete, lead to unintended emergent behaviors, or be incorrect. Additionally, interpretation of requirements during implementation adds an additional chance of error.

Chapter 3 presented natural language (NL) specifications for multiple pacemaker operating modes. We begin our journey to safely using AI in a pacemaker here with the translation of the NL specification into a formal language that can be tested for correctness. The first step is defining a formal language, Duration Calculus (Section 4.1.4), capable of expressing the intricate and precise pacemaker timing rules. Each formal language can be converted into an equivalent automaton, a feature that will be exploited to model and test the resulting formal pacemaker specification. Unfortunately, highly expressive formal languages suffer from the curse of that expressiveness where conversion to automata results in infinite or nondeterministic state machines. Before the pacemaker formal specification can be created a constrained version of DC will be introduced that restores decidability and determinism. Once this preliminary work is completed, formal VVI and DDD pacemaker specifications will be presented along with a validation of several complex DDD requirements to show formal specification correctness.

6.1 A History of Formal Pacemaker Specifications

Since Boston Scientific released their simple dual chamber pacemaker specification [86] in 2007, there have been many attempts to express the document's requirements in formal languages. Bonfanti [21] released a wide survey of formal methods usage in medical devices covering applications in modeling, verification, and validation. In the survey, modeling included papers focused on translation of the pacemaker specification into a formal language. Exemplar papers dedicated to formal pacemaker specification are [60, 89, 105]. These papers strive to translate the original English language pacemaker requirements into one of several formal languages. Each paper presents a unique method for dealing with the lack of real-time structure in the chosen formal language.

Gomes and Oliveira [60] developed a formal pacemaker specification in Z [92]. Z is a specification language built on formal proofs of propositional and predicate logic with no inherent construct for representing hard real-time limits and arbitrary, stochastic time periods. As presented in their paper, Gomez and Oliveira were able to specify and verify a simple DDD pacing system using the Z proof package ProofPower-Z. Their specification is incomplete as it lacks fundamental aspects of safe pacing, namely refractory and blanking timing and rate response. Blanking and refractory timing may be stochastically retriggered multiple times in the same V-V interval pushing their termination time beyond the end of the V-V interval. This is a difficult requirement to generate in many formal languages. The authors closed their paper looking at a derivative language that combines Z with CSP (Communication Sequential Processes) to overcome this issue. Additionally, their software sense amplifier for detecting intrinsic heart beat is incorrect as it relies solely on the amplitude of what appears to be an unfiltered incoming heart signal. Referring back to Section 3, the sense amplifier looks for amplitude change within a specific frequency range to avoid detection of non-cardiac activity and is thus looking for time and duration in a filtered signal.

Méry and Singh's work [105] is one of the few examples of formalizing the pacemaker specification [86] into a specification. Their work was written in the EVENT B modeling language and the RODIN tool set was used to generate and verify proof obligations. EVENT B provided

structure for arbitrary clocks, freeing the authors from the real-time limitation of predicate logic. It was found during the verification process that static proof analysis was not sufficient to assure proper operation. Functional modeling was required to identify deadlocks and logic failures, which could be attributed to missing actions in the model's event descriptions. In the end they succeeded in specifying and verifying formal requirements for simple versions of each type of single and dual chamber pacemaker. They did not extend the work to cover the more complex safety features such as PVC detection and PMT. The rate response implementation for addressing the need for accelerated pacing rate due to sensor detected activity was a simple toggle between nominal lower rate and a fixed higher sensor rate whereas an actual pacemaker will gradually increase the current pacing rate for a more comfortable change in heartbeat.

Larson [89] formalized the pacemaker specification [86] through creation of a BLESS based specification, an extension to the AADL modeling language. AADL is a modeling language for defining systems as a collection of connected components. A feature of AADL is the enforcement of interface types between components. BLESS adds formal properties, states, time, and guards to these interfaces to allow for generation of formal proof assertions. The BLESS implementation appears to be a full translation of the pacemaker requirements but there was no verification performed on the final model to show functionality or accuracy of the translation.

In addition to the attempts highlighted from Bonfanti's survey, Jiang [74] translated the pacemaker specification [86] into state machine models compatible with the UPPAAL model checking tool. Translating straight to automaton freed their effort from the constraints of representing time in a formal language. Their results took the translation process the furthest of any published effort and, with their model, they were able to show verification of complex pacemaker features such as pacemaker mediated tachycardia (PMT) detection and termination.

6.2 Decidable Subclasses of Duration Calculus

In the introduction to DC in Section 4.1.4 the undecidability of unconstrained DC was discussed and some different methods of constraining DC were provided that restored decidability.

Presented here are two additional methods for restoring decidability that translate easily into automata, Event-Triggered DC (EDC) and Time-Triggered DC (TDC). These logics restrict timing to a single observed event (EDC) or time point (TDC) from which all timing is measured. An additional restriction removes the $\int P$ measurement operator leaving $\ell \bowtie c$ and $\sum P$. These restrictions recover decidability resulting in the restricted EDC[ℓ] and TDC[ℓ] subclasses of DC.

EDC[ℓ] and TDC[ℓ] compile into event-clock [46] and integer-reset timed automata [83] respectively, which are determinizable subclasses of timed automata by removing the state space explosion of potentially infinite stopwatches running and halted simultaneously. These subclasses can then be used in searching for the optimal strategy of a PTA [68, 132] using an integer value only digital-clock abstraction [68].

6.2.1 Event-Triggered Duration Calculus

In many real-time systems, a specific event marks the start of a series of timed steps or is used to coordinate multiple parallel timed operations that must all complete in a time span relative to the starting event. It can be argued that many timing relationships in real-time systems are local in nature where future events must occur within a relative, and countable time span from the initial triggering event. Taking advantage of this characteristic, restricting the scope of DC's time measurement functions allows for describing local timing relationships while recovering decidability.

As an example, in the case of the pacemaker, a ventricular paced event triggers the VA Interval and also the various blanking and refractory periods. If an atrial intrinsic event occurs during the VA Interval, it is important to know which, if any, atrial related periods the intrinsic event falls into based on the time since the ventricular paced event.

Such measurements can be accomplished by restricting ℓ specifically to an event expressed as $\ell \bowtie^{\bullet}_{B} c$ where ℓ is the time measured since the most recent occurrence of the proposition B. Formally, given a timed word σ, an event B at the point interval $[b, b]$, and a future point interval

$[e,e]$ then $(\sigma,[b,e]) \models \ell \bowtie^{\bullet}_{B} c$ iff

$$\exists b < e \text{ s.t. } (\sigma,[b,b]) \models B \text{ and } (\sigma,[b,e]) \models (\lceil \neg B \rceil \wedge \ell \bowtie c) \text{ where } c = e - b.$$

Definition-3: Event-Triggered Duration Calculus

Given a set $P \in AP$ of finite set of atomic propositions, we define the syntax of EDC[ℓ] formula as follows:

$$\begin{aligned} P &::= \bot \mid \top \mid x \in AP \mid P \wedge P \mid \neg P \\ D &::= \lceil P \rceil \mid \lceil P \rceil^{\bullet} \mid D \wedge D \mid \neg D \mid D^{\frown} D \mid_{EDC[\ell]} \\ M_{EDC[\ell]} &::= \ell \bowtie^{\bullet}_{B} c \mid \sum P \bowtie c \text{ where } \bowtie \in \{<, \leq, =, \geq, >\} \end{aligned}$$

6.2.1.1 Event Clock Automata

When introduced in Chapter 4.1.2, it was noted that a stopwatch automaton (PSA) is not determinizable because due to the many freedoms on starting and stopping clocks on each action, stochastic transitions, and freedom to reset clocks on each transition. After the formal definition of the PSA, subclasses were presented where stochasticity and stopwatches were progressively removed finally resulting in the Timed Automaton (TA).

For a TA, each transition is defined by an action, evaluation of any pertinent clock constraints, and a fixed set of clocks that are to be reset. The current valuation of each clock represents the passage of time since the action when it was last reset. Flexibility in selecting which actions cause clocks to reset provides expressive power. Such flexibility unfortunately leaves the TA nondeterministic as each state transition may have two or more transitions for each action based on different sets $Y \in X$ of clocks to be reset. Alur, Fix, and Henzinger [9] returned determinism through a subclass of the TA where clocks resets are limited to association with specific actions or events creating the Event Clock Automaton (ECA) [8, 57].

For an ECA, given a finite alphabet Σ, C_{Σ} is the set of event clocks over the alphabet where $C_{\Sigma} = H_{\Sigma} \cup P_{\Sigma}$. The historical clock $x_a \in H_{\Sigma}$ of symbol $a \in \Sigma$ represents time since the last occurrence of a and has the valuation of $v(x_a) \in \{t_i - t_j, \bot\}$ where t_i is the i^{th} position in the timed

word $\overline{w} = (a_0, t_0)(a_1, t_1)\ldots(a_i, t_i)$, $j \leq i$, and t_j is the largest preceding j where $a_j = a$. If there is no prior occurrence of a then $v(x_a) = \perp$. The prophecy clock $y_a \in P_\Sigma$ of symbol a represents a's expected next occurrence time and has the valuation of $v(y_a) \in \{t_k - t_i, \perp\}$ where t_k is the predicted position of a's next occurrence or $\perp$ if a is never expected to occur again. Taken together, the valuation function $v \in V(C_\Sigma) \mapsto \{Rplus, \perp\}$. A clock constraint φ_j is a boolean conjunction of clock valuations $\wedge_i [v(x_i \in C_\Sigma) \bowtie c_i]$ where the constraint $c_i \in Q_{\geq 0}$ and $\bowtie \in \{\leq, \geq\}$. If at t_j the clock valuation function v_j evaluates to true against the constraint φ_j then $v_j \models \varphi_j$.

Definition-4: Event Clock Automaton

An Event Clock Automaton (ECA) is a tuple $A = (Q, q_i, \Sigma, \delta, \alpha)$ where

- Q is a finite set of locations,
- q_i is a set of starting locations $q_i \in Q$,
- Σ is the alphabet,
- δ is a finite set of edges $\delta \subseteq Q \times \Sigma \times \varphi \times Q$, and
- α is a set of accepting locations $\alpha \subseteq Q$.

Duration Calculus is a logic that references time to past events. As such, when converting EDC to an ECA, only historical clocks (H_Σ) are relevant and prophecy clocks (P_Σ) are removed. This reduces an ECA to an Event-Recording Automaton (ERA). ECA will be continued to be used in this document with the understanding that the prophecy clocks are not used.

6.2.1.2 Translating EDC[ℓ] to ECA

Translation from EDC[ℓ] to ECA is a three step process. Algorithms 6 translates from EDC to DDC by replacement of all measurement statements. Algorithm 7 converts the DDC logic to an automaton, while Algorithm 8 completes the conversion process creating an ECA.

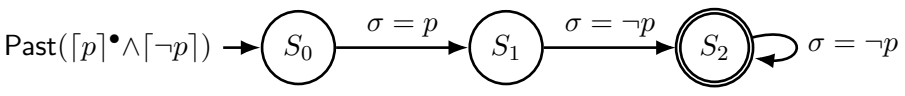


Figure 6.1: DFA translation for Past

Algorithm 6 ElimMeasure(D):Eliminate Measurement Constructs from D, $D \in$ EDC

Input: EDC formula $D(\varphi_1, \ldots, \varphi_n)$ over Var, with measurement constructs $\varphi_1, \ldots, \varphi_n$, each $\varphi_i = B_i \curvearrowleft M_i$.

Output: DDC+Past formula D' with no measurement constructs, over Var'=$Var \uplus \{E_1, \ldots, E_n\}$

1: **for** each φ_i **do**
2: Introduce a new propositional variable E_i
3: Replace φ_i by $W_i = \lceil E_i \rceil^\bullet \wedge (\lceil B_i \rceil^\bullet \wedge \lceil \neg B_i \rceil)$
4: **end for**
5: $D' = D[\varphi_i \leftarrow W_i]$

Event-Triggered Duration Calculus bases timed measurements on the most recent occurrence of integral time point events symbolized as $\lceil P \rceil^\bullet$. Given a time point (b, b) when P holds, $\mathsf{Past}(\lceil P \rceil^\bullet \wedge \lceil \neg P \rceil)$ will be used as shorthand to assert that there is some time point (e, e) where $b < e$ such that $\lceil \neg P \rceil$ holds over (b, e). Thus, with the timed symbols $(\sigma, [b, b])$ and $(\sigma, [e, e])$:

$$(\sigma, [e, e]) \models \mathsf{Past}(\lceil P \rceil^\bullet \wedge \lceil \neg P \rceil) \text{ iff}$$

$$\exists b < e, \quad (\sigma, [b, b]) \models P_i, \quad (\sigma, [b, e]) \models \lceil \neg P_i \rceil.$$

Starting with an EDC[ℓ] formula D, the first step in translation to an ECA requires removal of the measurement constructs $\varphi = \ell \bowtie^\bullet_P c$. Replacing each constraint with $\lceil E_i \rceil^\bullet \wedge \mathsf{Past}(\lceil P_i \rceil^\bullet \wedge \lceil \neg P \rceil)$ where E_i is a fresh propositional variable to be used later as a witness for satisfaction of the removed constraint. This results in a simpler formula over a larger number of propositional variables $AP' = AP \uplus \{E1, \ldots, En\}$. This transformation of D to D' by removing time measurements results in a Discrete DC (DDC) formula enhanced with the Past macro or DDC+Past. The transformation can be seen in Algorithm 6 where line 1 loop over each measurement constraint while lines 2 and 3 introduce E_i and make the Past substitution. This process is illustrated with a pacemaker example.

Example 6-1

Consider the VVI pacemaker upper rate requirement. For a patient with an upper rate setting of 130bpm, ventricular paces may not occur on a period shorter than 462ms (e.g. $60s/min \div 130bpm$). This leads to the measurement constraint $\varphi_p = \ell \geq^\bullet_p 462$ over $AP = \{p\}$ where p represents the atomic predicate for a Ventricular Pace (VP) and time is measured in milliseconds. The requirement

in DC would look like

$$D(\varphi) = (\lceil p \rceil^{\bullet} \frown \lceil p \rceil \rightarrow \top \frown \ell \geq_p^{\bullet} 462).$$

Using the fresh variable E_1, Algorithm 6 returns the following formula:

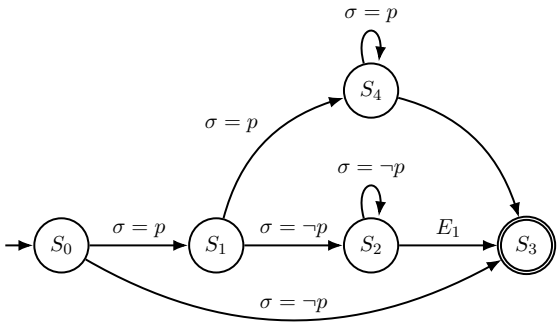
$$D' = (\lceil p \rceil^{\bullet} \frown \lceil \neg p \rceil) \rightarrow \top \frown [\lceil E_1 \rceil^{\bullet} \wedge \mathsf{Past}(\lceil p \rceil^{\bullet} \wedge \lceil \neg p \rceil)] \, \text{over} AP = \{p, E_1\}.$$

■

The next step in the translation process is converting the resultant formula D' from DDC+Past into a deterministic finite automaton (DFA). Algorithm 7 is an inductive process starting by translating each of the atomic DC statements into their equivalent DFA and then combining the resulting DFAs based on the binary DC operators to compose the final DFA. The DC to DFA equivalents are shown in Figures 4.4a (Range), 4.4b (Point), 4.4c (Chop), 6.1 (Past). The final step for Algorithm 7 is to remove time from $A(D')$ by removing the time stamps from the words of the original language $w \in L(D')$ where the language of the resultant automaton is $L(A(D')) = Untime(L(D'))$.

Example 6-2

Continuing from Example 3–1, Algorithm 7 now translates the DDC+Past into a DFA. The VVI requirement is that after a VP, another VP may not occur any sooner than 462ms, thus, an integral point p must be followed by a length of time where $\neg p$ holds. Algorithm 7 first translates $\mathsf{Past}(\lceil p \rceil^{\bullet} \wedge \lceil \neg p \rceil)$ into $A(D_1)$ per Figure 6.1. The witness variable E_1 translates into $A(D_2)$ per Figure 4.4b and the $\frown$ operator concatenates $A(D_1)$ and $A(D_2)$ per Figure 4.4c. Finally, the error path where $\neg p$ never occurs is added completing the translation and returns the automaton $A(D')$:



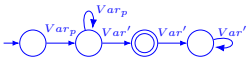
The S_4 to S_3 edge is a don't care as the S_4 is a fail state due to p continuing to hold. ■

With the untimed DFA $A(D')$ given by Algorithm 7, Algorithm 8 proceeds with the creation the desired ECA. First, for each measurement constraint $\varphi_i = \ell \bowtie^{\bullet}_{P_i} c_i$ a clock x_i is added at line 2. Algorithm 8 then proceeds to cycle through each edge of $A(D')$ and either resets or tests each clock based on if the associated witness variable E_i is pertinent. When E_i is not pertinent to the transition, lines 5–8 check if the current symbol matches the starting event P. For each constraint where word a matches the starting event, it's associated clock x_i is added to the list of clocks to reset — the measurement cannot start until the start event no longer holds. If the start event is no longer asserted then lines 10–15 add the time constraint. If the witness E_i holds then the measurement constraint is added to the edge in lines 10–11, otherwise the negative of the constraint is added in lines 13–14. This completes the creation of the ECA.

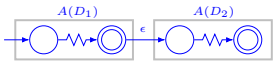
Algorithm 7 Formula2Aut(D'):Convert DDC+Past formula D' without measurements to automata $A(D')$

Input: DDC+Past Formula D' over Var' with no measurement constructs.
Output: Automaton $A(D')$ s.t. $L(A(D'))$=$Untime(L(D'))$.

1: **if** $D' = \lceil P \rceil^{\bullet}$ **then**
2: $A(D')$= 

3: Determinize $A(D')$
4: **end if**
5: **if** $D' = \lceil P \rceil$ **then**
6: $A(D')$= 

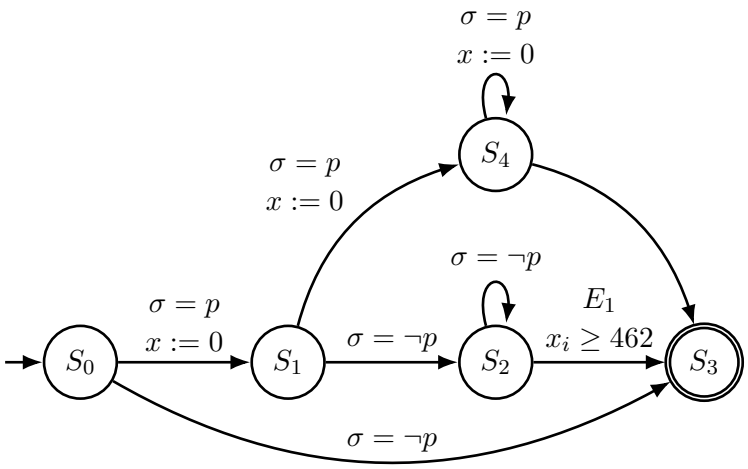
7: Determinize $A(D')$
8: **end if**
9: **if** D'=Past($\lceil B \rceil^{\bullet} \wedge \lceil \neg B \rceil$) **then**
10: $A(D')$= 

11: Determinize $A(D')$
12: **end if**
13: **if** $D' = D_1 \frown D_2$ **then**
14: $A(D')$= 

15: Determinize $A(D')$
16: **end if**
17: **if** $D' = D_1 \wedge D_2$ **then** $A(D) = A(D_1) \cap A(D_2)$
18: **end if**
19: **if** $D' = \neg D_1$ **then** $A(D) = \overline{A(D_1)}$
20: **end if**

Example 6-3

Continuing from Example 3–2, Algorithm 8 begins by creating a timer x for the constraint $\varphi \geq 462$. It then cycles through each edge. For edges $(S_0, S_1), (S_1, S_4), (S_4, S_4)$ x is reset as p, representing a VP, is still holding and, per the requirement, a new VP event cannot occur until 462ms after p is no longer true. E_1 is checked on the edge (S_2, S_3) so Algorithm 8 adds the constraint test to the edge. For edges $(S_1, S_2), (S_2, S_2)$ p is not asserted after p held. This is the time the requirement intends to measure and x is allowed to run (count up). Finally, for edge (S_0, S_3) p never held and thus clock x is undefined and has the value of $\perp$. The final DFA is:



6.2.2 Time-Triggered Duration Calculus

In time-triggered systems, event occurrences are coordinated through use of a single master or global clock. Kopetz [83] recognized that a global clock is not always realizable depending on system architecture where local clocks may have some skew that would make verification difficult. To overcome this constraint, Kopetz described a method reducing dense time to sparse time consisting of active and silent periods capturing near simultaneous events into time blocks or intervals.

Kopetz's concept of a global time reference over a real-time system motivates the time-triggered subclass of DC. With TDC, measurements become referenced to globally integral points of time (tic), $M_{TDC} = \bowtie^{\bullet}_{tic} c$. Formally, given a timed word σ, a time point interval $tic = [b, b]$,

Algorithm 8 Aut2ECA(A): Convert automaton A over Var' to event clock automata $ECA(A)$

Input: Automaton A over Var', such that $L(A) = Untime(L(D'))$ for DDC+Past formula D' over Var'.
Output: $ECA(A)$ such that $L(ECA(A)) = L(D')$.

for each $\varphi_i = \ell \bowtie^{\bullet}_{B_i} c_i$ **do**
  Introduce a clock variable x_i
end for
for each transition $q \xrightarrow{a} q'$ in A, $a \subseteq Var'$ **do**
  Introduce guard ψ and reset $Y \subseteq \{x_1, \ldots, x_n\}$ as:
  $\psi \leftarrow true$
  if $a \models B_i$ **then**, $Y = Y \cup \{x_i\}$
  end if
  for $1 \leq i \leq n$ **do**
    if $(a \models E_i) \wedge \varphi_i = (\ell \bowtie^{\bullet}_{B_i} c)$ **then**
      $\psi = \psi \wedge (x_i \bowtie c)$
    end if
    if $(a \not\models E_i) \wedge \varphi_i = (\ell \bowtie^{\bullet}_{B_i} c)$ **then**
      $\psi = \psi \wedge \neg(x_i \bowtie c)$
    end if
  end for
end for
Replacing all transitions of A, ECA(A) is obtained.

and a future point interval $[e, e]$ then $(\sigma, [e, e]) \models \ell \bowtie^{\bullet}_{tic} c$ iff

$$\exists b < e, \sigma, [b, e] \text{ s.t. } (\sigma, [b, e]) \models \ell \bowtie c \text{ where } c = e - b$$

Definition-5: Event-Triggered Duration Calculus

Given a set $P \in AP$ of finite set of atomic propositions, we define the syntax of EDC[ℓ] formula as follows:

$$\begin{aligned} P &::= \bot \mid \top \mid x \in AP \mid P \wedge P \mid \neg P \\ D &::= \lceil P \rceil \mid \lceil P \rceil^{\bullet} \mid D \wedge D \mid \neg D \mid D^{\frown} D \mid_{TDC[\ell]} \\ M_{TDC[\ell]} &::= \ell \bowtie^{\bullet}_{tic} c \mid \sum P \bowtie c \text{ where } \bowtie \in \{<, \leq, =\geq, >\} \end{aligned}$$

Like EDC[ℓ] translates into a decidable ECA, TDC[ℓ] translates into an Integer Reset Timed Automaton (IRTA). The basics of IRTA will be presented, followed by the translation method for TDC[ℓ] to IRTA.

6.2.2.1 Integer Reset Timed Automata (IRTA)

IRTA [137] are a subclass of stopwatch automata which are determinizable and closed under Boolean operations making them well behaved. Similarly to an SWA, IRTA transitions take the form $T = (q, a, Y, \varphi, q')$ where $Y \subseteq C$ is the set of clocks to be reset on the transition and φ is the set of clock constraints. Different from an SWA, all clocks have a fixed rate $\gamma = 1$ and γ can be safely dropped from the IRTA transition tuple. By tradition for IRTA, x is typically used to represent both clocks $x \in C$ and the clock valuation $v(x)$. The remainder of this introduction to IRTA will adhere to this nomenclature when it is obvious from the description if the clock or its valuation are being referenced.

If $|Y| \neq 0$, constraints $\varphi \in \Phi(C)$ must be of the form $(x = c) \wedge \varphi$ where $x, c \in N_0$ and $x \in Y$. The requirement $x = c$ with their valuations as natural numbers ensures all clock resets to occur on integral time while no clock has valuation in Q thus assuring decidability. The constraint φ remains as described in SWA and ECA $\varphi = x \bowtie c$ where $\bowtie \in \{\leq, <, >, \geq, \varphi \wedge \varphi\}$.

Definition-6: Integer Reset Timed Automaton

An Integer Reset Timed Automaton (IRTA) is a tuple $A = (Q, q_i, \Sigma, C, E, F)$ where

- Q is a finite set of locations,
- q_i is a set of starting locations $q_i \in Q$,
- Σ is the alphabet of the timed language,
- C is the set of all clocks,
- E is the set of all transitions $E \subseteq Q \times Q \times \Sigma \times \Phi(C) \times 2^C$, and
- F is the set of accepting locations $\alpha \subseteq Q$.

6.2.2.2 Translating TDC[ℓ] to IRTA

The translation of TDC[ℓ] to a decidable automaton follows the same basic three steps as translating EDC[ℓ] to an ECA — replacement of all time measurement constraints with an ob-

Algorithm 9 ElimMeasure(D) : Eliminate Measurement Constructs from D, $D \in \text{TDC}[\ell]$

Input: TDC[ℓ] formula $D(\varphi_1, \ldots, \varphi_n)$ over AP, with measurement constructs $\varphi_1, \ldots, varphi_n$, each $\varphi_i = \ell \bowtie^{\bullet}_{\mathtt{tic}} c_i$.

Output: DDC formula D' with no measurement constructs, over AP'=$AP \uplus \{E_1, \ldots, E_n\}$

1: **for** each φ_i **do**
2: Introduce a new propositional variable E_i
3: Replace φ_i by $\lceil E_i \rceil^{\bullet}$
4: **end for**
5: $D' = D[\varphi_i \leftarrow \lceil E_i \rceil^{\bullet}]$

server (Algorithm 9), construction of an equivalent automaton (Algorithm 7), and insertion timing constraints into the resultant automaton on the edges with their associated observer (Algorithm 10).

Like EDC Algorithm 6, Algorithm 9 replaces each time measurement constraints with an observer predicate E_i, thus converting the TDC formula to a DDC D. Unlike the EDC process, Past is not required as all IRTA time constraints are referenced to a specific integral time point and therefore the need to track an event that held and then no long was asserted is unnecessary. Translation of D to an automaton uses the same rules as for an EDC formula and Algorithm 7 is again used to return the automaton $A(D)$.

Again, like the EDC translation process, the final step is to insert the time constraints into $A(D)$. Algorithm 10 begins by introducing a clock variable x_i for each witness E_i that was added by Algorithm 9 (Line 2). The next step creates structure to assure each state has an assured integer time transition. Line 4 creates a clock z that counts from 0 to 1 and is then reset. Then lines 5–6 modify each state q_i into a timing pair by adding an additional state q'_i. The transitions between these two states is identical with a guard $z = 1$ and an action $z := 0$ reseting z. Next, lines 8–12 duplicate all transitions based on receipt of a symbol $q \xrightarrow{a} q'$ to create a transition with integer time ($z = 0$) and another with non-integer time ($0 < z < 1$). These transitions are created with the set of clocks to reset Y_i empty. The Y_i for each transition will be assigned in the next step.

To maintain integer time on measurement clocks x_i, they may only be reset on transitions where $z = 0$ and this is performed in lines 13–18 where Y_i for each state is compiled (line 14) and then assigned (line 17). The final step is to add the time constraints to those edges containing

Algorithm 10 Aut2IRTA(A): Convert automaton A (from Algorithm 7) over AP' to integer reset timed automata $IRTA(A)$

Input: Automaton A over $AP' = AP \uplus \{E_1, \ldots, E_n\}$, such that $L(A)$=$Untime(L(D'))$ for DDC D' over AP'.
Output: $IRTA(A)$ such that $L(IRTA(A)) = L(D')$.

1: **for** each $\varphi_i = \ell \bowtie^{\bullet}_{\mathtt{tic}} c_i$ **do**
2: Introduce a clock variable x_i
3: **end for**
4: Add an extra clock variable z to track global integral time
5: **for** each location q in A **do**
6: Add a loop on q which checks z=1 and resets z to 0
7: **end for**
8: **for** each transition $q \xrightarrow{a} q'$ in A, $a \subseteq AP'$ **do**
9: Replace with 2 transitions $(q, a, \emptyset, z{=}0, q')$ and $(q, a, \emptyset, 0{<}z{<}1, q')$
10: $(q, a, \emptyset, z{=}0, q')$ takes places at globally integral times
11: $(q, a, \emptyset, 0{<}z{<}1, q')$ takes places at globally non integral times
12: **end for**
13: **for** $1 \leq i \leq 2^n - 1$ **do**
14: $Y_i \leftarrow$ nonempty subset of $\{x_1, \ldots, x_n\}$ s.t. $Y_i \neq Y_j$ for $i \neq j$.
15: **end for**
16: **for** each transition $(q, a, \emptyset, z = 0, q')$ **do**
17: replicate into 2^n transitions $(q, a, Y_1, z{=}0, q'), \ldots, (q, a, Y_{2^n}, z{=}0, q')$,
18: **end for**
19: **for** each transition (q, a, Y, ψ, q') **do**
20: Replace as follows
21: **if** $a \models E_i$ **then**, $Y \leftarrow Y \cup \{x_i\}$
22: **end if**
23: **for** $1 \leq i \leq n$ **do**
24: **if** $(a \models E_i)$ **then** $\psi = \psi \wedge (x_i \ op \ c)$
25: **end if**
26: **if** $(a \not\models E_i)$ **then** $\psi = \psi \wedge \neg(x_i \ op \ c)$
27: **end if**
28: **end for**
29: **end for**
30: Replacing all transitions of A, IRTA(A) is obtained.

observers E_i. First, line 21 checks if a satisfies the current observers. If so, the associated clock x_i is added to Y to be reset. Next, if a satisfies E_i then the time constraint is added to the list of time constraints for this transition. If a does not satisfy then the negated version of the time constraint is added to the transition constraint list. Once this is completed, IRTA(A) is returned.

6.2.3 A Spoonful of Sugar — Derived Operators for DC

Deriving syntactic sugar for DC can make dense formulas easier to read. With that mind, here are some useful derived operators that will be used when DC formula are presented later.

(1) Usage of $\lceil X \rceil^\bullet$ and $\lceil X \rceil$ is extended over Boolean combinations of predicates as follows:

$$\lceil X \bowtie Y \rceil^\bullet \stackrel{\text{def}}{=} \lceil X \rceil^\bullet \bowtie \lceil Y \rceil^\bullet$$

$$\lceil \neg X \rceil^\bullet \stackrel{\text{def}}{=} \neg \lceil X \rceil^\bullet$$

$$\lceil X \rceil \stackrel{\text{def}}{=} \Box(\lceil X \rceil^\bullet \frown \top)$$

$$\lceil X \rceil^d \stackrel{\text{def}}{=} (\ell{=}d \wedge \lceil X \rceil) \frown \top$$

where $\bowtie$ is some logical binary operator and X and Y are Boolean formulas over predicates.

(2) Operator $A \hookrightarrow_d B$ encodes that A triggers B for precisely d time units. If A occurs again before d time then period d is restarted. B could hold indefinitely should A continue to occur without d time between assertions:

$$A \hookrightarrow_d B \stackrel{\text{def}}{=} \Box\Big((\ell \geq d+1) \wedge (\lceil A \rceil^\bullet \frown \top) \Rightarrow (\ell = 1) \frown \lceil B \rceil^d\Big) \wedge$$
$$\Box\Big((\ell \geq d+1) \wedge \lceil \neg A \rceil^d \Rightarrow (\ell{=}d+1) \frown \lceil \neg B \rceil^\bullet \frown \top\Big).$$

(3) Operator $A \stackrel{C}{\leftrightarrow} B$ encodes that C holds between events A and B, i.e.,

$$A \stackrel{C}{\leftrightarrow} B \stackrel{\text{def}}{=} \Box(\ell \geq 1 \wedge (\lceil A \wedge \neg B \rceil^\bullet \frown true) \Rightarrow (\ell{=}1 \frown waitFor(B, C))) \wedge$$
$$\Box(\ell \geq 1 \wedge (\lceil B \rceil^\bullet \frown true) \Rightarrow (\ell{=}1 \frown waitFor(A, \neg C)))$$

where $waitFor(X, Y)$ states that Y will hold until X holds for $\ell = 1$

$$waitFor(X, Y) \stackrel{\text{def}}{=} \lceil \neg X \wedge Y \rceil \vee$$
$$(\lceil \neg X \wedge Y \rceil \frown \ell{=}1 \frown \lceil X \wedge Y \rceil^\bullet \frown true) \vee$$
$$(\lceil X \wedge Y \rceil^\bullet \frown true).$$

(4) Operator $A \curvearrowright B$ encodes that B is true when A transitions from $\top$ to $\bot$, i.e.,

$$A \curvearrowright B \stackrel{\text{def}}{=} \Box(\lceil A \rceil^{\bullet} \frown (\ell = 1) \frown \lceil \neg A \rceil^{\bullet} \Leftrightarrow (\ell = 1) \frown \lceil B \rceil^{\bullet}) \frown true.$$

(5) Operator $A \stackrel{B?}{\hookrightarrow}_d C$ encodes C will be true after A and holds until either a delay of time d or arrival of B,

$$A \stackrel{B?}{\hookrightarrow}_d C \stackrel{\text{def}}{=} (A \hookrightarrow_d X) \wedge (A \stackrel{Y}{\leftrightarrow} B) \wedge \lceil (X \wedge Y) \Leftrightarrow C \rceil$$

where X and Y are intermediate signals used to compose the behavior of C.

(6) Operator $(A \wedge B) \stackrel{\neg B}{\hookrightarrow} C$ encodes that C will be true when B turns false after both A and B are true,

$$(A \wedge B) \stackrel{\neg B}{\hookrightarrow} C \stackrel{\text{def}}{=} (B \curvearrowright X) \wedge ((A \wedge B) \stackrel{Y}{\leftrightarrow} X) \wedge (Y \curvearrowright C)$$

where X and Y are intermediate signals with X representing when B becomes false and Y representing the period when B holds.

6.3 VVI Pacemaker DC Specification

Given the definitions of the constrained DC subsets of EDC and TDC it is possible to write a formal specification of a VVI pacemaker as seen in Table 6.1. The right hand column of the table is a cross reference to one or more of the original NL requirements as given in Section 3.4 and highlights one of the difficulties in requirement translation. Natural language syntax allows for freedom in how a need is presented and may include parts of multiple requirements in a single statement. Enforcing formal syntax requires the translation process to reconsider how functionality is distributed in the NL specifications. Here, the VVI NL requirements were translated manually into those seen in the table.

VVI pacing is fairly simple. In the absence of intrinsic activity, the pacemaker should pace at the lower rate. The first requirement, *lowvp*, states this case where an instantaneous ventricular pace ($\lceil VP \rceil^{\bullet}$) occurring implies that the lower rate interval must have passed ($(\ell =^{\bullet}_{VP} (pace - 1))$)

Table 6.1: VVI Pacemaker : TDC[ℓ] specification

Rule		**TDC[ℓ] specification**	Req.
$lowvp$	=	$\lceil VP \rceil^{\bullet} \frown true \implies ((\ell =^{\bullet}_{VP} (pace - 1)) \wedge \Box \lceil \neg VP \rceil)$	VVI-1 VVI-4 VVI-5
$onlyref$	=	$(\ell =^{\bullet}_{\mathtt{tic}} pace) \wedge \Box((\ell =^{\bullet}_{\mathtt{tic}} ref) \wedge \Box \lceil \neg VS \wedge \neg VP \rceil \implies$ $((\ell =^{\bullet}_{\mathtt{tic}} (ref + 1)) \frown \lceil \neg VS \rceil^{\bullet}))$	VVI-2 VVI-7
$pace\_req$	=	$((\ell =^{\bullet}_{\mathtt{tic}} pace) \frown \lceil VP \rceil^{\bullet})$	VVI-1
no_pace_req	=	$(\ell =^{\bullet}_{\mathtt{tic}} pace - 1) \wedge (\Box \lceil \neg VP \rceil)$	VVI-1 VVI-3
$repeatvp$	=	$\Box((\lceil VP \rceil^{\bullet} \frown true) \wedge onlyref \implies pace\_req)$	VVI-4
$vstrigger$	=	$[((\lceil VS \rceil^{\bullet} \frown true) \wedge onlyref) \implies pace\_req] \wedge [(\lceil VS \rceil^{\bullet} \frown true \implies no\_pace\_req)]$	VVI-10
$repeatvs$	=	$\Box[((\ell =^{\bullet}_{\mathtt{tic}} ref) \wedge \Box(\lceil \neg VS \wedge \neg VP \rceil)) \implies ((\ell =^{\bullet}_{\mathtt{tic}} ref + 1) \frown vstrigger)]$	VVI-9 VVI-11 VVI-13
$boot$	=	$((onlyref \implies pace\_req) \wedge (\ell =^{\bullet}_{\mathtt{tic}} pace - 1 \wedge \Box \lceil \neg VP \rceil))$	VVI-6
Specification:	=	$lowvp \wedge repeatvp \wedge repeatvs \wedge boot$	

with no ventricular paces occurring during that interval ($\Box \lceil \neg VP \rceil$). That requirement is then combined with $pace_req$ which simply states that any interval duration of lower rate ($pace$) occurs then it is always followed by a ventricular pace.

6.4 DDD Pacemaker Specification

In Chapter 3.4 the specification for a simple DDD pacemaker was given based on the Boston Scientific pacemaker specification [86], while removing extraneous features such as communication and more complex features such as rate response for simplicity of demonstration and verification. In this section, the requirements of that specification will be translated to DC for later use in creating a reward generator.

6.4.1 Dual Chamber Pacemaker Signals, Constants, and Timers

The pacemaker DC specification consists of formulas interpreted over signals indicating event occurrences and indicators that periods and intervals are active as described in Chapter 3.4. The names described here are the DC representation of those signal as depicted in Figure 3.11a. Naming

will follow the convention that constants are given exclusively lower case names while signal and timer names have one or more capital letter in their name.

Input and Output Signals. Inputs to the pacemaker are events specifying intrinsic heart activity and are limited to VS (ventricular sense) and AS (atrial sense) signals for activity in the ventricle and atrium respectively. The output pacemaker signals consist of the ventricular and atrial paced events VP and AP respectively.

Time Interval Constants. Figure 3.11a showed the various periods and intervals generically without specifying actual duration. Duration is based on the patient's current condition and type of cardiac disease. As a patient's heart rate changes through activity and rest, durations lengthen or shorten somewhat proportionally and each value is defined in the pacemaker specification. Assigning a name to each constant simplifies the reading of the DC formula with the understanding that constants can be continually changing based on heart activity.

- `lrl` (lower rate limit) is the longest allowed time between a sensed or paced ventricular event and the next paced event.
- `url` (upper rate limit) is the shortest time possible between two consecutive ventricular paced or sensed events.
- `pav` (paced atrial-ventricular interval) is the maximum time between an atrial pace and the next scheduled ventricular pace.
- `sav` (sensed atrial-ventricular interval) is the maximum time between a sensed atrial beat and the next scheduled ventricular pace.
- `va` (ventricular-atrial interval) is the maximum time between a sensed or paced ventricular event and the next scheduled atrial pace.
- `pvab`, `pavb`, `pvvb`, `paab`, `svab`, `savb`, `svvb` and `saab` are the duration of the various blanking periods. The names are coded per Table 3.3.

- `pvvr`, `pvar`, `paar`, `svvr`, `svar` and `saar` are the duration of the various refractory periods. The names are coded per Table 3.3. It should be noted, as shown in Figure 3.11a, there is no ventricular refractory period (*i.e.* savr, pavr) during the AV interval.

- `safe` is the length of the safety pacing period following an atrial paced or sensed event starting an AV interval.

Timers. For the pacemaker to correctly decide when to pace, it needs to know where in the cardiac cycle the heart currently is. Intervals and periods making up the cardiac cycle shown in Figure 3.11a can be implemented as a timer whose state is tested for *running* or *expired* allowing the pacemaker to determine the next correct action based on the combination of timers currently running. When translating the pacemaker specification in to DC, these timers become predicates that hold for an amount of time. For example, the predicate $PVVR$ represents the post ventricular event ventricular refractory period and holds true for a maximum duration of `pvvr`. The pacemaker specification allows $PVVR$ to be truncated in some situations. The timer predicates are:

- $PAAB$, $PAVB$, $SAAB$, $SAVB$, $PVAB$, $PVVB$, $SVAB$, $PVVB$ are the blanking periods after each sensed or paced event.

- $PAAR$, $SAAR$, $PVAR$, $PVVR$, $SVAR$, $SVVR$ are the refractory periods after each sensed or paced event. As noted earlier, there is no ventricular refractory period after an atrial event.

- PAV or SAV holding indicates the AV interval is currently active following either a paced (PA) or sensed (SA) event.

- VA indicates the the AV interval is currently active following either a paced (PV) or sensed (SV) event.

- LRL indicates that the lower rate interval timer is still active.

- URL indicates that the upper rate interval timer is still active.

Derived Signals. Some pacemaker requirements are reliant on the status of other requirements and the following signals are used to broadcast requirement status. These signals are not necessarily related to timers but give further context to system state when events occur.

- $ExPVAR$ is a special predicate that holds when $PVAR$ requires extension due to an intrinsic ventricular beat occurring when $PVAR$ is currently asserted.
- $EPAV$, $ESAV$ and EVA indicate the end of the PAV, SAV and VA intervals respectively as transitions from 1 to 0.
- ASA and VSA hold when a sensed atrial or ventricular sensed event occurs outside of blanking.
- ASN and VSN hold when a sensed atrial or ventricular sensed event occurs outside of refractory.
- $NoAS$ holds in the absence of an atrial sense after a non-refractory ventricular sensed event until an ASA event is seen.
- `scheduledVP` is asserted when the VP is the scheduled pace occurring at the end of the AV interval.
- `lateVP` is asserted when the scheduled VP must be delayed due to upper rate holdoff requirements. The delayed pace shall occur when this signal is no longer asserted.
- `safeVP` indicates a safety ventricular pace VSP occurred.
- $BOOT$ is a special predicate used as a trigger to start the system.
- VO is a special signal indicating when optimal permissive ventricular pacing is allowed. Permissive pacing will be describe in the following case study.

Table 6.2: DDD Pacemaker Case Study : TDC[ℓ] Requirements

Rule		**TDC[ℓ] Requirement**	Req.
$boot$	=	$\lceil BOOT \rceil^{\bullet} \vee \lceil BOOT \rceil^{\bullet} \frown \ell = 1 \frown \lceil \neg BOOT \rceil$	
$vblankperiod$	=	$(VP \hookrightarrow_{\texttt{pvvb}} PVVB) \wedge (AP \hookrightarrow_{\texttt{pavb}} PAVB) \wedge (VSA \hookrightarrow_{\texttt{svvb}} SVVB) \wedge (ASA \hookrightarrow_{\texttt{savb}} SAVB)$	DDD-9 DDD-21
$vblank$	=	$\lceil VS \wedge \neg PVVB \wedge \neg PAVB \wedge \neg SVVB \wedge \neg SAVB \Leftrightarrow VSA \rceil$	DDD-2 DDD-7 DDD-15
$ablankperiod$	=	$(VP \hookrightarrow_{\texttt{pvab}} PVAB) \wedge (AP \hookrightarrow_{\texttt{paab}} PAAB) \wedge (VSA \hookrightarrow_{\texttt{svab}} SVAB) \wedge (ASA \hookrightarrow_{\texttt{saab}} SAAB)$	DDD-1 DDD-8 DDD-14 DDD-20
$ablank$	=	$\lceil AS \wedge \neg PVAB \wedge \neg PAAB \wedge \neg SVAB \wedge \neg SAAB \Leftrightarrow ASA \rceil$	DDD-7
$vrperiod$	=	$((VP \vee VSA) \hookrightarrow_{pvvr} PVVR) \wedge ((VP \vee VSN \vee (ASA \wedge PVAR)) \hookrightarrow_{\texttt{pvar}} PVAR)$	DDD-10 DDD-11 DDD-22 DDD-23 DDD-29 DDD-32
$nonvr$	=	$\lceil \neg PVVR \wedge VSA \Leftrightarrow VSN \rceil$	DDD-30
$arperiod$	=	$(AP \vee ASN) \xleftrightarrow{PAAR} (VP \vee VSN)$	DDD-3 DDD-16
$arlen$	=	$\lceil \neg PAAR \wedge \neg PVAR \wedge \neg ExPVAR \wedge ASA \Leftrightarrow ASN \rceil$	DDD-4 DDD-17
pav	=	$(AP \xhookrightarrow{VSN?}_{\texttt{pav}} PAV) \wedge (ASN \xhookrightarrow{VSN?}_{\texttt{sav}} SAV)$ $\wedge((PAV \wedge \neg VSN) \frown EPAV) \wedge ((SAV \wedge \neg VSN) \frown ESAV)$	DDD-5 DDD-18 DDD-31
$vastart$	=	$(VSN \vee ESAV \vee EPAV \vee BOOT) \xhookrightarrow{ASN?}_{\texttt{va}} VA \wedge ((VA \wedge \neg ASN) \frown EVA)$	DDD-23 DDD-24 DDD-27
pvc	=	$(VSN \xleftrightarrow{NoAS} ASA) \wedge ((NoAS \wedge VSN) \hookrightarrow_{\texttt{expvar}} ExPVAR)$	DDD-12
$lrlurl$	=	$((VP \vee VSN) \hookrightarrow_{\texttt{lrl}} LRL) \wedge ((VP \vee VSN) \hookrightarrow_{\texttt{url}} URL)$ $\wedge((AP \vee ASN) \hookrightarrow_{\texttt{lrl}} ALRL) \wedge ((AP \vee ASN) \hookrightarrow_{\texttt{url}} AURL)$	DDD-33
$apsched$	=	$\lceil (EVA \wedge \neg ASN \wedge \neg AURL) \Leftrightarrow scheduledAP \rceil$	DDD-13 DDD-25
$aurlholdoff$	=	$((EVA \wedge \neg ASN) \wedge AURL) \xhookrightarrow{\neg AURL} lateAP \wedge \lceil (scheduledAP \vee lateAP) \Leftrightarrow AP \rceil$	DDD-33
$vpsched$	=	$\lceil ((EPAV \vee ESAV) \wedge \neg VSN \wedge \neg URL) \Leftrightarrow scheduledVP \rceil$	DDD-6 DDD-19
$vpholdoff$	=	$(((EPAV \vee ESAV) \wedge \neg Vother) \wedge URL) \xhookrightarrow{\neg URL} lateVP$	DDD-33
$vsafety$	=	$(AP \hookrightarrow_{\texttt{safe}} SAFE) \wedge (VSN \wedge SAFE) \xhookrightarrow{\neg SAFE} safeVP$ $\wedge \lceil (scheduledVP \vee lateVP \vee safeVP) \Leftrightarrow VP \rceil$	DDD-26
lrl	=	$\lceil VP \Rightarrow LRL \rceil$	DDD-1
url	=	$\lceil VP \Rightarrow \neg URL \rceil$	DDD-28 DDD-33

6.4.2 The DC Pacemaker Specification

Given these timer, constant, and signal definitions it is possible to translate the simple dual chamber (DDD) pacemaker requirements given in Section 3.4. The DC requirements are shown in Table 6.2. Like the VVI DC requirements table, each NL requirement was manually translated while each resulting DC formula in the table is traced back to the originating requirements where a DC formula may implement part or all of one or more requirement. The DDD requirements are significantly more complex than those for the VVI pacemaker. The complete specification for the DDD pacemaker is the concatenation of all the requirements.

To improve requirement readability in the table the DC syntactic sugar has been used as macros to capture the various timing relationships in a more visual nature. As an example, the atrial refractory period, *arperiod*, starts on an atrial event ($AP \vee ASN$) and holds ($\xleftarrow{PAAR}$) until either an intrinsic ventricular beat or the next scheduled ventricular pace ($VP \vee VSN$), whichever occurs first. Likewise, upper rate holdoff ($upholdoff$) is defined such that if, at the end of the PAV or SAV where no ventricular event has occurred ($(EPAV \vee ESAV) \wedge \neg Vother$) and the upper rate interval is still running then when the upper rate limit does complete($(\wedge URL) \xhookrightarrow{\neg URL}$), it triggers a late or held off ventricular pace ($lateVP$).

6.4.3 Pacemaker Specification Verification

The first step to doing something novel with a pacemaker is verifying the basic specification produces a model that works as intended. While complete verification of the specification is beyond the scope of this thesis, general correctness can be shown through two of the more complex pacemaker features, Ventricular Safety Pacing (VSP) (DDD-29, DDD-30, DDD-31) and Upper Rate Holdoff (DDD-32, DDD-33).

Formal verification is typically done by converting the formal specification φ to an automaton model $\mathcal{M}$ from which one can show that the model satisfies the specification (e.g. $\mathcal{M} \models \varphi$) by traversing the automaton. In the VVI formal specification of Table 6.1 the final statement is the

concatenation of all the DC formula to create a single statement. This process impractical for the much larger DDD specification due to state space explosion followed by a difficult minimization effort. Instead, the DDD pacemaker verification model consists of individual automata for each DC formula, which are run separately. This requires the modeling algorithm to order the automata with respect to their inter-dependency to assure those generating inputs for other automata execute first and to prevent any circular logic where two automata depend results of the other.

The model automata are created in a two-step process. First, the DC formula are translated to monadic second order (MSO) statements using a tool called DCVALID [43]. The MONA tool [67] converts the MSO statements to automata where are walked using traces hand designed to specifically create conditions triggering the expected pacemaker response.

Validation Case 1 (Ventricular Safety Pacing). Ventricular Safety Pacing (VSP) requires recognizing early intrinsic ventricular beats that fall during the VSP window that started with the preceding atrial intrinsic beat or pace and scheduling an early ventricular pace to occur when the window expires. See Figure 3.12 for a diagram of the basic VSP logic.

Figure 6.2 is created from a run of the model where an intrinsic beat is inserted within the VSP window. Many aspects of the pacemaker specification can be verified in this diagram. Starting with the VA signal we can see that when the VA interval expires, an atrial pace (AP) is correctly generated. The AP starts both the PAV and VSP window (SAFE), seen as rising edges of their signal traces. This is followed by a ventricular sense (VS) that occurs while SAFE is asserted, thus falling in the VSP window and scheduling a safety VP. The scheduled safety VP (safeVP) can be seen to occur at time point 20 synchronized with the fall of (SAFE).

The final signal to verify is LRL. It starts low due to how the model handles initial startup. The first AP starts the next LRL interval and it can be seen that LRL is never deasserted indicating that the timing between all atrial-to-atrial beats, intrinsic or paced, came before LRL expired. This meets the requirement to never pace below the lower rate.

The Figure 6.2 trace verifies correct operation of the VSP logic along with the logic transi-

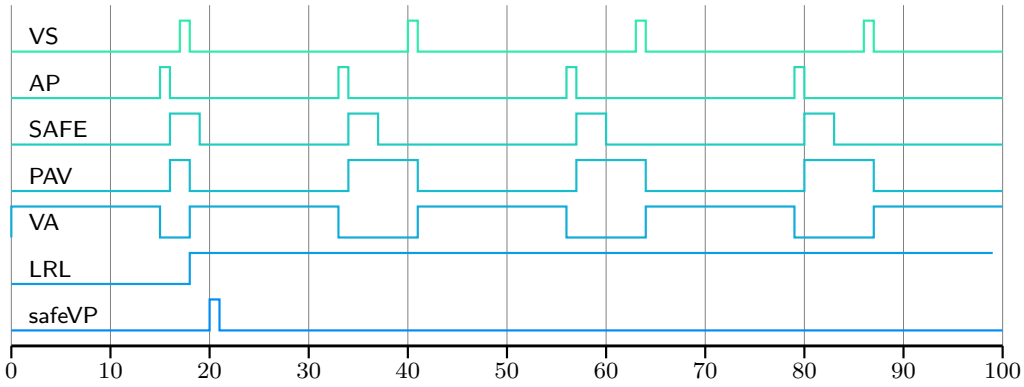


Figure 6.2: Ventricular Safety Pacing Internal Pacemaker Signals

tioning between the VA and AV intervals and proper LRL response.

Validation Case 2 (PVC, LRL, and Upper Rate Hold-off). The second validation case implements the contrived URL hold off scenario depicted in Figure 3.13 where the time from a VS to the next scheduled VP would violate the Upper Rate Limit while the next AP would violate LRL. In this scenario, the VP must be held off until the URL time is satisfied yet the next VA interval must start upon the termination of the AV interval with no delay to meet LRL. Additionally, the test begins with a PVC-R to test correct rejection of the intrinsic ventricular beat but with associated restart of the refractory periods.

The test begins with testing PVC-R response. Starting with an AP, the PAV interval begins. The end of the PAV is followed immediately by a VP and the start of the refractory periods PVVR, PVAR and the upper rate timer URL. This is correct operation.

Next, a VS occurs at $t = 24$ while PVVR is still asserted. This is a PVC-R and should not be interpreted as a valid ventricular intrinsic. This behavior can only be inferred in the Figure by the response to the following AS. Had the PVC-R been incorrectly interpreted, PVAR would have been restarted and the AS would have fallen into the atrial refractory period. Instead, the AS triggers the start of SAV, the correct response. There is an error in the PVC-R response however in that PVVR should have been restarted and from the figure it can be seen that its length does not vary.

The AS fell sufficiently close to the end of PVAR that the next scheduled VP would violate the

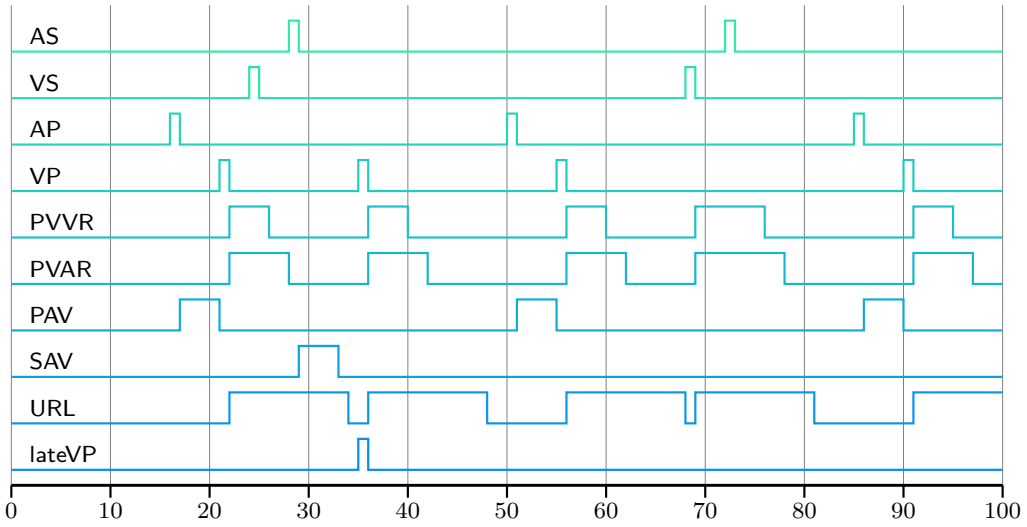


Figure 6.3: Upper Rate Holdoff Internal Pacemaker Signals

URL. The model recognizes this with the signal lateVP indicating that the nest VP may not happen until URL completes and indeed, the next VP coincides with lateVP. This is correct operation.

Aside from the lack of reset on PVVR in the PVC-R test, these validation tests show the model is operating properly and this error will not affect the following case study.

6.5 Conclusions

In this chapter it was shown that it is possible to write a formal language specification for a complex real-time system. The duration and time measurement features of Duration Calculus were instrumental in expressing pacemaker timing constraints. Unconstrained DC, however, does not translate to an automaton suitable for use as a model. By restricting time to only positive integers and forcing all time measurements to refer to the same point in time or event, translation to a decidable, deterministic automaton was possible.

Using the constrained versions of DC specifications were presented for both the single chamber VVI pacemaker and the more complex DDD pacemaker. Using a simple translation process an automata DDD model was created allowing for verification of two complex requirements.

Chapter 7

Correct Reward Machines For RL Pacemaker Training

Using RL to synthesize a controller agent through rewards for correct actions was described in Section 4.2.1. The accuracy of the reward, generated by a rewards machine, is critical to training a viable and safe RL agent. The machine directs the agent's learning by providing positive and negative feedback as rewards for actions taken by the agent. If the machine provides incorrect feedback, the agent is not able to determine an optimal policy and may learn undesired behavior. Getting the rewards machine correct is a difficult task. This chapter expands on that basic understanding to address the challenge of creating a reward machine that is guaranteed correct.

7.1 Creating Reward Machines

Reward machine design for complex or stochastic systems can be difficult. Incorrect emphasis of intermediate rewards can skew the learning agent to maximize near-term actions at the expense of long-term goals. This is best showcased in Amodei and Clark's paper [35] where the RL policy learned for an arcade boat racing game emphasized collecting local bonus points for near-term rewards over the long-term reward for completing the race. This behavior is an example of reward hacking [129], the optimization of an inferior policy due to an imperfect rewards function or machine. A reward machine may be imperfect for many reasons including noise in the input signal, coding errors such as buffer overflows the RL agent learns to exploit, the RL agent learning to influence the rewards machine through its actions. This chapter addresses incorrect design and coding errors.

A rewards machine can be represented as a Büchi automaton where reward is given on each

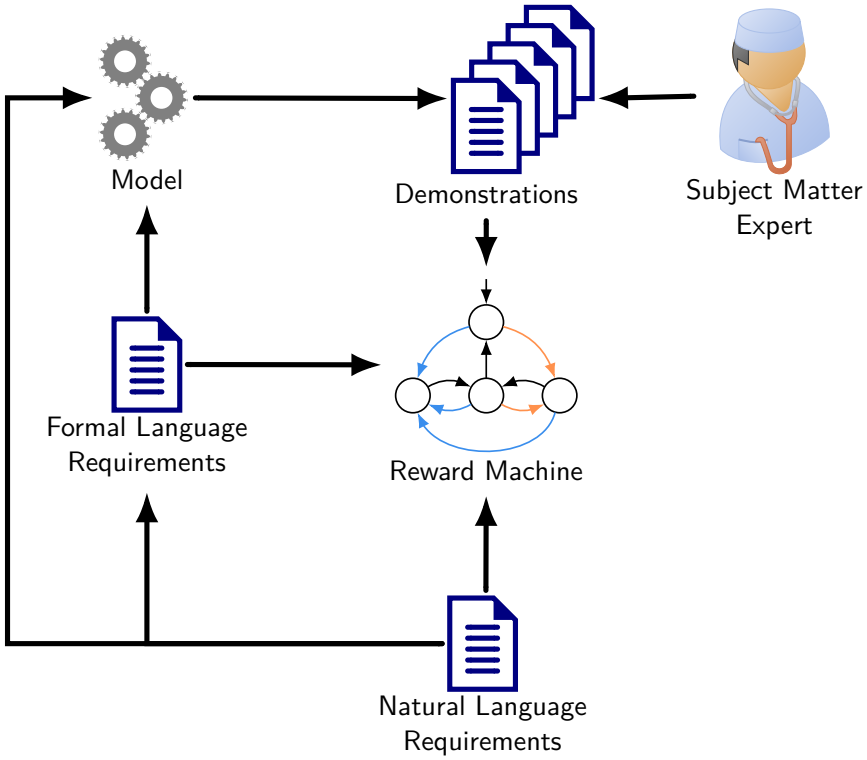


Figure 7.1: Ways to Create an RL Rewards Machine

visit to an accepting state or penalty for visiting a unaccepting state. It does not encode the entire controller the agent is learning. Instead it encodes the rules that should lead to acceptable environmental behavior. In the case of a pacemaker, a rewards machine automaton would encode the rules for when the RL based pacemaker must pace.

There are many ways to create a rewards machine as depicted in Figure 7.1. The most common method is manual translation from a specification or problem description directly to an automaton. This translation process will inherit the same translation errors as described earlier when translating to formal languages. Alternatively, Chapter 6 showed how to translate a specification into formal language requirements readily convertible to an automaton which can operate as a rewards machine. This will be demonstrated through the remainder of this chapter. Also depicted is reward machine creation from demonstrations and will be demonstrated in a later chapter.

7.2 Rewards Machines From Formal Specification

The goal of a learning RL agent is to synthesize a controller that satisfies a specification based on interaction with an environment represented as unknown Probabilistic Timed Automaton (PTA). Formally, given a set of atomic propositions AP, a PTA $\mathcal{T} = (Q, I, \Sigma, X, E, \delta, F)$, a labeling function $\mathcal{L} : Q \to AP$, and specification φ, the controller synthesis problem is to compute a strategy $\pi \in \Pi_{[\![\mathcal{T}]\!]}$ that maximizes the probability of satisfaction of φ. This is problematic however as the synthesis problem must learn the PTA's probability distribution $\delta(q, a)$ over a potentially uncountable infinite state space from repeated sampling of the environment. The undecidability of synthesizing this PTA over continuous time is a corollary of Chaochen, et al [29]. The single chamber pacemaker case study presented later in this chapter shows how RL's creative exploration addresses this undecidability.

Two case studies will be used to show the utility of EDC[ℓ] and TDC[ℓ] in practice for modeling real-time systems and the ability of RL to derive the optimal policy of an unobservable MDP without learning the MDP's transition probabilities when the RL agent's actions are constrained by a reward machine created from formal requirements. The equivalence between an MDP and a PTA was given in Chapter 4.1.2.

In each study, a system is specified formally with DC requirements then converted to a timed automata through the EDC[ℓ] (TDC[ℓ]) to ECA (IRTA) processes described earlier in Chapter 6. The resultant automaton is then used as the RL agent reward machine guiding the RL agent to take acceptable actions that yield the optimal policy for the case study. The environment will be modeled as an unobservable and stochastic MDP.

7.2.1 Grid World

Grid world is a classic environment used for demonstrating reinforcement learning techniques. Given an $x \times y$ grid with a fixed starting and goal location and multiple traps, the optimal policy finds the route that returns the highest reward. In each square of the grid, the agent selects a

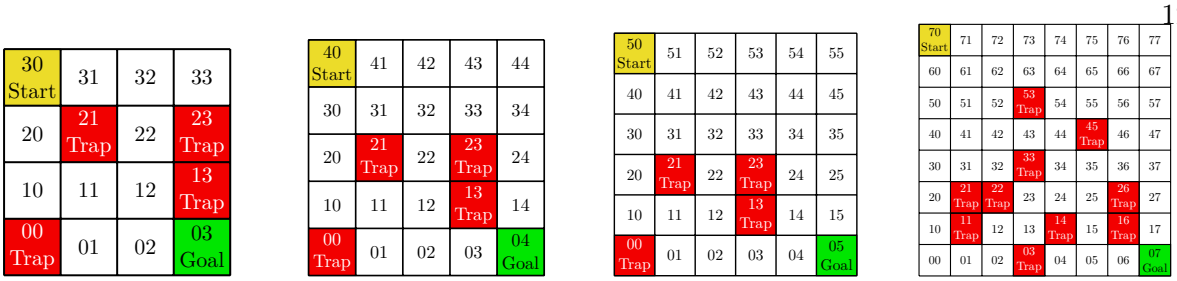


Figure 7.2: Frozen Lake grids used in case study

cardinal direction. If the selected move would cause the agent to leave the grid, the agent does not move for that action. Play ends when the agent moves either into a trap or the goal. Should the last move result in a trap location, the RL agent is given a negative reward. Grid world environments are not stochastic and are easily solvable.

Frozen lake is a stochastic extension on the grid world environment where the agent's actual next state is probabilistic. When the agent selects an action, there is a probability p of moving in the desired direction and $(1-p)/2$ of moving in either of the adjacent directions. For all of the frozen lake case studies listed in Table 7.1 except for study 3b, $p = 1/3$. The challenge of finding the optimal policy was gradually increased by adding additional rules such as the agent may not select the same action twice in a row. Taking a different action each time may lead to unsolvable grids or less optimal policies. To overcome this possibility, an additional action was added to allow the agent to simply *wait* for one turn allowing the agent to repeat the same action with an interspersed *wait* between them. The grids used for this case study are shown in Figure 7.2. Overall, four variations of the case study were performed. The differences and gradually increasing challenge is shown in Table 7.1.

The specification for frozen lake case study 2 is given in Table 7.2 and consists of six rules written in DC. First, *pick_one* states that only one action may be selected on each turn. $no\_conflict$ states that only one action may occur for any reason in an increment of time. $\lceil reach\_goal \rceil$ states that the agent must always reach the goal without entering a trap location. *no_consecutive_req*

and *no_con_wait* define that after each action taken, a period of 1 must follow where that action is not taken. *no_trap* states that a trap location should never occur. The complete case study 2 specification is the concatenation of these six requirements.

The process for each case study began with conversion of the DC requirements into EDC[ℓ] (TDC[ℓ]) automata. MUNGOJERRIE [108] was then used to find an optimal strategy using RL and compare the result against the theoretical optimal strategy. MUNGOJERRIE is a OpenAI gym-like [24] framework of tools designed for experimenting with finite MDP and stochastic game reward agent design over objectives described in ω regular languages and LTL specifications. For the frozen lake case studies, MUNGOJERRIE was used to calculate the probability of satisfaction for each case study using probabilistic model checking (MC), then RL to learn an approximate optimal policy, and compare the two results.

Table 7.3 shows the results of the various experiments run. For each grid size and rule set, the table shows the episode size and time required to find an approximate optimal policy and then the comparison of that policy with the expected (MC). The results show that RL can attain a reasonable approximation in most cases, though MC still returns a better overall result. Generally it can also be seen, and not unexpectedly, that as the grid size and complexity of the rules increase, the length of training required increases substantially. The 8×8 case study could only find a solution for the simplest rule set.

Analyzing the policies learned by the RL lead to some interesting observations of the reasoning encoded in the results and where it can become handicapped. The most important rule that it

Table 7.1: Frozen Lake Case Study Requirements

Study	Requirements
1	Standard Frozen Lake with no modifications
2	Adds "delay" action to mimic then no move is made; The agent may not take any action consecutively
3a	Adds time limit; Agent must complete within 60 time units
3b	Same as 3a except probabilities changed to 1/4:1/2:1/4

Table 7.2: Frozen Lake Case 2 : EDC[ℓ] specification

Rule		**EDC[ℓ] specification**
$pick\_one$	=	$\lceil n \vee e \vee s \vee w \vee wait \rceil^{\bullet}$
$no\_conflict$	=	$(\lceil n \rceil^{\bullet} \Rightarrow \lceil \neg e \wedge \neg s \wedge \neg w \wedge \neg wait \rceil^{\bullet}) \wedge (\lceil e \rceil^{\bullet} \Rightarrow \lceil \neg n \wedge \neg s \wedge \neg w \wedge \neg wait \rceil^{\bullet}) \wedge$ $(\lceil s \rceil^{\bullet} \Rightarrow \lceil \neg n \wedge \neg e \wedge \neg w \wedge \neg wait \rceil^{\bullet}) \wedge (\lceil w \rceil^{\bullet} \Rightarrow \lceil \neg n \wedge \neg e \wedge \neg s \wedge \neg wait \rceil^{\bullet}) \wedge$ $(\lceil wait \rceil^{\bullet} \Rightarrow \lceil \neg n \wedge \neg e \wedge \neg s \wedge \neg w \rceil^{\bullet})$
$\lceil reach\_goal \rceil$	=	$\Box(\neg goal) \frown true \frown \lceil goal \wedge \neg trap \rceil^{\bullet}$
$no\_consecutive\_req$	=	$\Box((\lceil n \rceil^{\bullet} \frown true \Rightarrow \ell =^{\bullet}_{n} 1) \frown \lceil \neg n \rceil^{\bullet}) \wedge (\lceil e \rceil^{\bullet} \frown true \Rightarrow (\ell =^{\bullet}_{e} 1) \frown \lceil \neg e \rceil^{\bullet}) \wedge$ $(\lceil s \rceil^{\bullet} \frown true \Rightarrow (\ell =^{\bullet}_{s} 1) \frown \lceil \neg s \rceil^{\bullet}) \wedge (\lceil w \rceil^{\bullet} \frown true \Rightarrow (\ell =^{\bullet}_{w} 1) \frown \lceil \neg w \rceil^{\bullet}))$
no_trap	=	$\Box \lceil \neg trap \rceil$
no_con_wait	=	$\Box(\lceil wait \rceil^{\bullet} \frown true \Rightarrow (\ell =^{\bullet}_{wait} 1) \frown \lceil \neg wait \rceil^{\bullet})$
Specification:	=	$pick\_one \wedge no\_conflict \wedge reach\_goal \wedge$ $no\_consecutive\_req \wedge no\_trap \wedge no\_con\_wait$

learned, based on the stochastic nature of the environment, was the nonintuitive selection of actions that limited the chances of entering a trap, even at the expense of the path's expediency through the environment. For example, the optimal path to avoid traps in the 4×4 grid of Figure 7.2a is:

$$S_{30} \xrightarrow{W} S_{20} \xrightarrow{W} S_{10} \xrightarrow{N} S_{11} \xrightarrow{S} S_{01} \xrightarrow{E} S_{02} \xrightarrow{S} S_{03}.$$

When in S_{20}, the safest move is west, off the board because there's no chance of ending in trap S_{21}. It does, however mean there's a one-third chance of moving back to S_{30} and getting stuck oscillating between S_{20} and S_{30}.

Case study 2 added the wait command and disallowed taking any action twice. The intent was to simulate some need to pause such as perhaps the agent was heating up when it moved and needed time to cool down. When the agent was allowed to take as many waits as desired, it used *wait* to prevent moving into a trap by permanently waiting and never attempting to move again. This behavior was seen when the agent had 'bad luck' and ended in S_{22} where there was always at least a one-third probability of ending in traps S_{21} or S_{23}. Changing the rules for this case study to disallow consecutive waits prevented this behavior and the path became identical to case 1 except for a *wait* action between each move.

Case study 3a then looked to see if there was a way to force the agent to take actions that

Table 7.3: Frozen Lake Case Study Results

Grid Size	Rule Set	Size of Product	Prob of Satisfaction MC	Prob of Satisfaction RL	Episode Length	Episode Number	RL Train Time(secs)
4×4	1	311	0.780	0.780	1000	300,000	84.0
4×4	2	877	0.780	0.776	1000	300,000	59.2
4×4	3a	138,286	0.388	0.313	1000	75,000,000	10,038.3
4×4	3b	138,286	0.454	0.382	1000	75,000,000	11,841.7
5×5	1	545	1.000	0.939	1000	300,000	147.3
5×5	2	1511	1.000	0.997	1000	300,000	72.9
5×5	3a	289,320	0.996	0.788	400	15,000,000	3501.2
6×6	1	820	1.000	0.947	2000	900,000	440.5
6×6	2	2270	1.000	0.982	2000	900,000	227.1
6×6	3a	431,655	0.999	0.876	400	15,000,000	3880.1
8×8	1	1424	1.000	1.000	2000	600,000	550.0
8×8	2	2984	0.230	0.000	1000	75,000,000	22.5

were less safe in order to complete within a bounded time limit. This was a significantly larger challenge as there were now 60 copies of each state to represent their value for each tick of the integer bounded time limit clock. As seen in the results, the number of episode steps and episodes increased substantially yet the agent only achieved 80–90% of the MC predicted satisfaction. This case was not run on the 8×8 due to excessive learning times.

The final case study looked to see if changing the action probabilities to favor the selected action would have any effect. Using only the 4×4 grid and a probability for taking the selected action of 0.5, the agent did increase its probability of satisfaction but continued to trail the MC probability by a margin similar to case 3a.

7.2.2 VVI Pacemaker

Pacing therapy provides an environment with fixed, real-time deadlines. Unlike grid world where the rules for when an action may be taken are based only on current state, a pacemaker's actions are related to spans of time from an arbitrary point. As an example, the lower rate

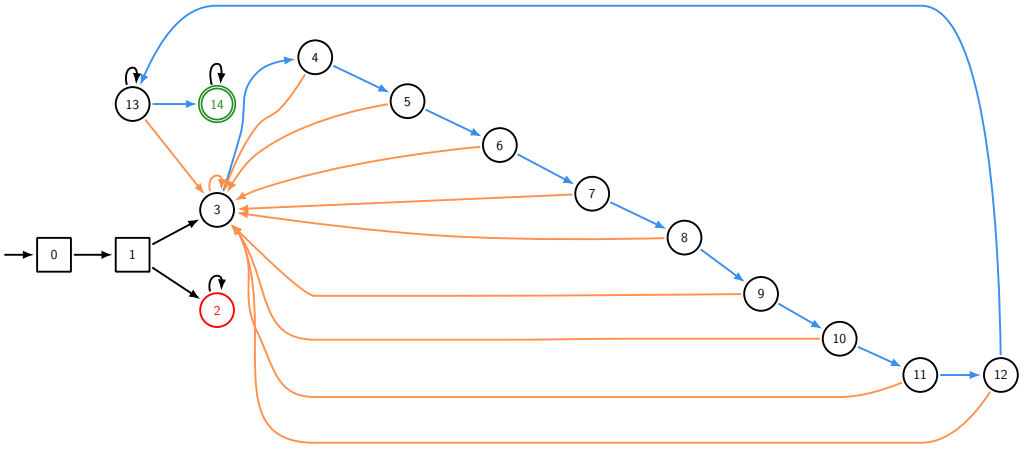


Figure 7.3: Unrolled Lower Rate Interval Automaton for lrl=10

requirement VVI-1 states that an amount of time equal to the lower rate must pass before a ventricular pace may occur. What triggered the start of the time span is irrelevant.

This case study attempts to learn the functionality of a simple VVI pacemaker (Figure 3.10b) by rewarding an RL agent based on correct operation. The reward machine implemented the VVI requirements given in Section 3.4. These were translated by hand to the TDC specification shown in Table 6.1. Blanking periods were not modeled for this case study. The same toolchain from the grid world case study was used for creating the pacemaker reward machine. This led to some compromise in the fidelity of the model due to limitations of the translation from requirements to automaton. The translation tool used an intermediate translation to Monadic Second Order (MSO) logic where all time periods were unrolled into a sequence of states, each state representing one clock tick. This constraint forced each pacemaker design to have fixed timing periods that could not be adjusted once the reward machine automaton was created. Figure 7.3 shows an example of how the equation

$$(\lceil VS \rceil^{\bullet} \vee \lceil vp \rceil^{\bullet}) \frown (\lceil \lceil (\neg VS \vee \neg VP) \rceil^{lrl}) \frown \lceil VP \rceil^{\bullet}$$

would be unrolled for an $lrl = 10$. This limitation restricts the pacemaker model to a single fixed rate of pacing and precludes modeling rate response where the pacing rate increases and decreases with patient activity.

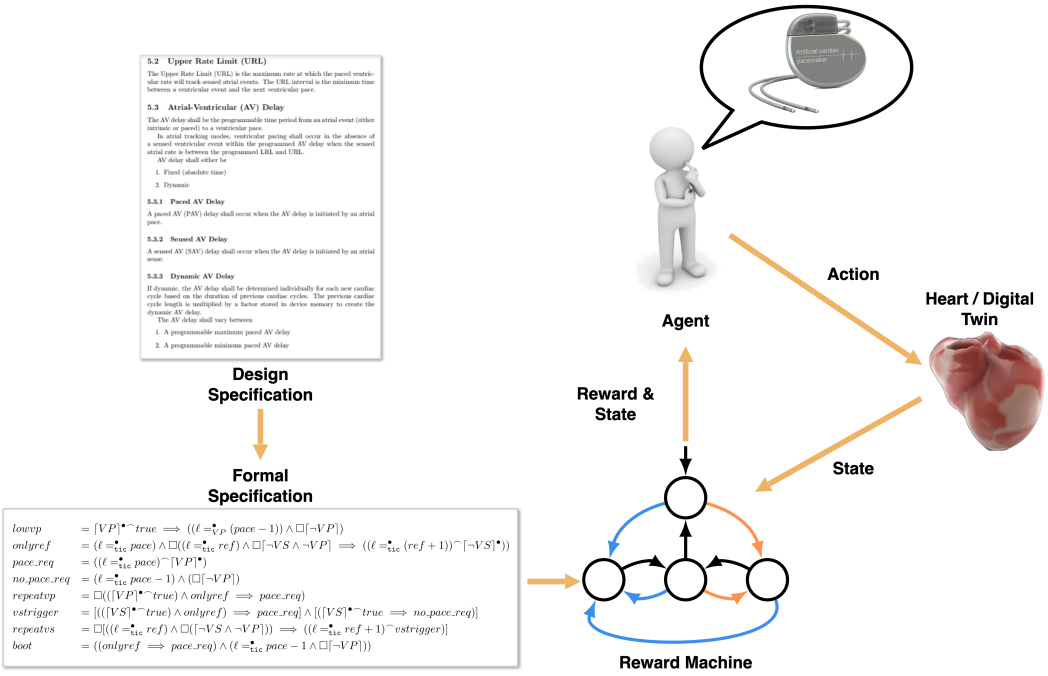


Figure 7.4: VVI Training Workflow

Training of the VVI pacemaker was performed as diagrammed in Figure 7.4 using the rewards machine automaton created from the VVI specification. The heart model was a simple event scheduler (Algorithm 11) where the next planned intrinsic beat was scheduled after each intrinsic beat or pacemaker paced event. For training, the heart was set to beat stochastically by picking a random time for the scheduled intrinsic event. A stochastic heart was used in order to expose the pacemaker to the most variety of times between intrinsic events. This is similar to how verification would be performed to assure pacemaker recognition of valid intrinsic paces versus refractory beats

Algorithm 11 stepHeart: Models a simple single chamber heart

Input: *pace* = Boolean indicating pacemaker paced.
Output: *event* = Boolean indicating intrinsic heart beat.

```
1: Increment Interval_Counter
2: if Interval_Counter = Event_Time ∧ Event_Scheduled then
3:     event = True
4:     ScheduleNewEvent()
5:     Interval_Counter = 0
6: end if
7: return event
```

Table 7.4: VVI Pacemaker Case Study Results

Case	Refractory Period	V-V Interval	Size of Product	Prob of Satisfaction MC	Prob of Satisfaction RL	Episode Length	Episode Number	RL Train Time(secs)
vvi–2–6	2	6	120	1.00	1.00	1000	100,000	3.78
vvi–3–8	3	8	270	1.00	1.00	1000	100,000	3.14
vvi–4–10	4	10	884	1.00	1.00	1000	250,000	6.65
vvi–5–14	5	14	2882	1.00	1.00	1000	5,000,000	171.3
vvi–6–16	6	16	4360	1.00	0.00	1000	10,000,000	8.85

though formal verification would precisely define when each intrinsic would occur.

Table 7.4 shows the five tests performed. Each test is labeled based on the unrolled timers. Thus, test vvi–2–6 had a refractory period of 2 and a V-V interval of 6 and for a lower rate of 60bpm (1000ms) each clock tick would represent 166.7ms.

The pacemaker had only two possible actions, *pace* and *wait*, and only one environmental input indicating if the ventricle intrinsically beat. The results in Table 7.4 show that Q-table learning can replicate the pacemaker specification given these allowed actions and environment input. This was an interesting result as the learning agent had to infer which intrinsic beats fell in the refractory interval and pace appropriately.

One can also see in the results table the effect of Q-table learning. Like the grid world case study, as the number of states in the automaton increased, the time to learn also increased to the point that test vvi–6–16 was unable to find a solution.

7.3 Rewards Machines From Expert Demonstrations

With Model Based development, introduced in Section 4.3, the designer builds a model representation of the intended design from a draft textual description. The model is then tested against the description and each is then updated to correct errors while adding additional detail. This iterative process continues until the model and description both represent the desired outcome. Standard tools are available to take the model forward through the design process with the expectation of minimal design errors. But would the process work if the model is a black box?

It's not uncommon when starting to design a replacement for an existing cyber physical system (CPS) to find the original documentation less than adequate. Changes were made during design not reflected in the specification. Documentation is lost or stored in a format no longer supported. It may be the original subject matter expert is no longer available. When this occurs, the only true source of requirements is the device itself [48]. By exercising the original device, traces of proper operation can be generated. By containing representation of the original design intent, traces from a model or existing product are an alternative method of specification translation. The focus of this chapter's case study is showing how these traces may be used to recreate the original functionality as a rewards machine.

As a further example of capturing design intent from traces, Chapter 5 described the process of analyzing surgical video. This is another type of black box where there is no established specification to begin with and there is variation in how the procedure can be performed. Analysis of each surgical video results in a trace representing one possible path for the surgical procedure. Like the true black box previously described, these traces contain the logic of the specification and can be used to train a product to perform the procedure or to score future traces of the procedure against the expert demonstrations contained in the original traces.

7.3.1 RL Training from Traces

Referring back to Figure 7.1, there are three basic methods to synthesis a rewards machine: from a natural language specification, from a formal language specification, and from expert demonstrations. Chapter 6 established that creation of a correct rewards machine from natural language requirements is difficult. The previous section demonstrated correct by design rewards machine synthesis from formal language requirements. This section addresses the remaining method of reward machine creation from demonstrations.

Demonstrations can be recorded as a sequential series of operations, steps, or evolutions of parameters. Each demonstration is one trace through the operation of the system to be controlled. Traces may be created through observation as described in Chapter 5, by executing a model derived

from a formal language specification or recorded runs of an original system. For completeness, the traces should demonstrate both good and erroneous operation though erroneous can be inferred as the set of all traces not represented by good traces.

Viewing a trace as a sequence of consecutive actions where one or more actions in the sequence may be wrong a reward machine can be a classifier neural network trained to identify is the trace has errors or not. There are many past examples of using a neural sequence architectures including electrocardiogram data [136, 31], brain-machine interfaces [32, 27], mobile networks [144], and various Internet-of-Things devices [96, 111]. This is the method used in the following case study.

7.3.2 Traces To Pacemaker

The case study for this chapter is straight forward and ultimately results in the same outcome as the Chapter 7 where a rewards machine trains an RL agent to operate as a pacemaker. The stakes are higher here though with the more goal of learning the complicated dual chamber pacemaker.

To begin, sets of good and bad traces need to be generated. While there are some cardiac waveform databases available [130, 10], we were required to build our own datasets from scratch. Many available waveform databases are electrocardiogram (ECG) based. This is a body surface signal with a different morphology than the intracardiac electrogram (EGM) waveform measured by an implanted pacemaker. EGM libraries like [10] contain analog waveforms measured at the pacemaker leads. Before such signals can be used by pacemaker pacing logic they are processed through a chain of analog and digital filters and comparators to remove noise sources such as muscle activation potential and breathing to isolate the actual intrinsic pacing pulse. Our research does not attempt to model this functionality and assumes this portion of the pacemaker has been properly configured to produce clean intrinsic activity markers.

7.3.2.1 Traces to Rewards

The trace generation environment for this case study consisted of the pacemaker automaton created and tested in Chapter 8 and a heart model that can exhibit healthy function or one of

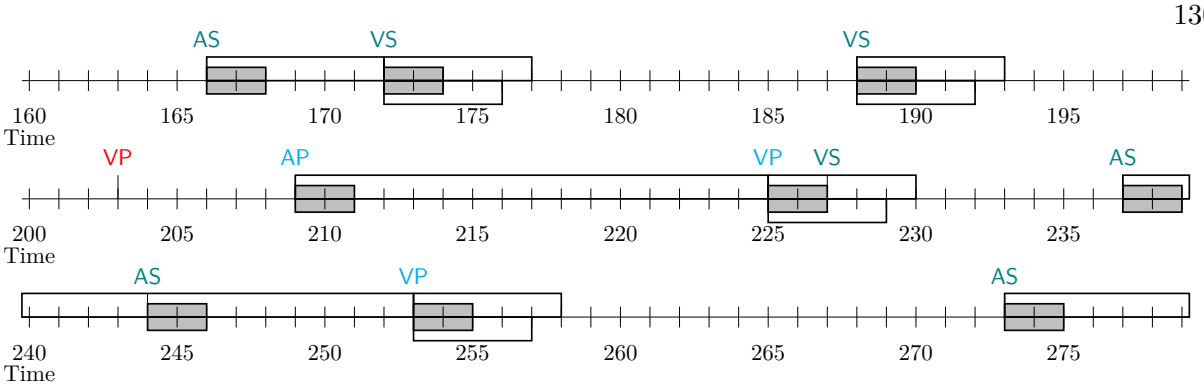


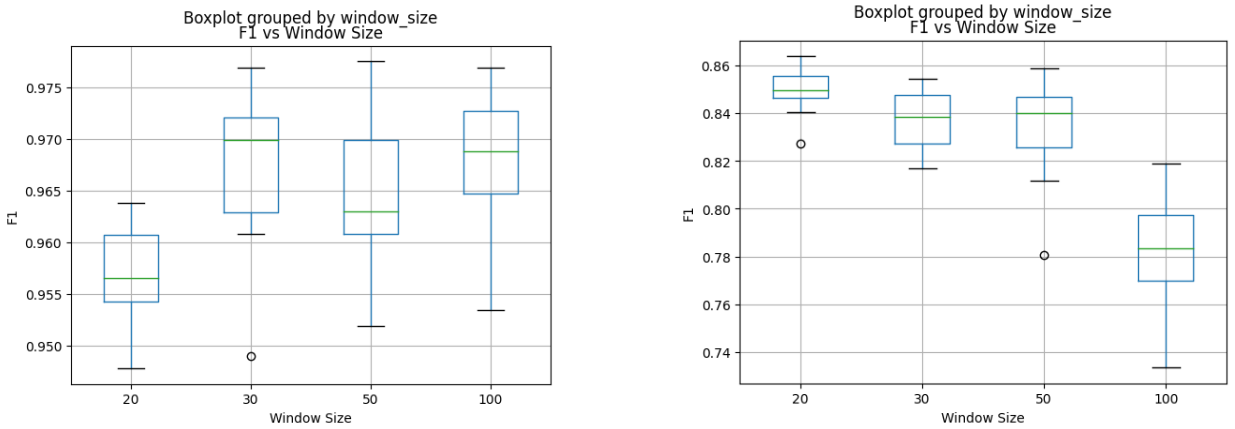
Figure 7.5: Example segment of a bad pacemaker trace with errant pace at time 203

several common cardiac arrhythmias. The arrhythmias modeled were: sick sinus with complete AV block, sinus arrest, premature ventricular contractions (PVCs), Mobitz II (3:2 heart block), and stochastic. In each heart mode the intrinsic rate was allowed to stochastically drift up and down so the RL agent couldn't key in specifically on the timing between paces. It would need to understand that as time between paces changes, so does the length of the blanking and refractory periods. Bad traces contained errors of omission, where a scheduled pacemaker generated pace should have occurred but did not, and extraneous, where additional pacemaker paces occurred in erroneous locations and the location and type of each error was randomly selected. A short portion of a trace is shown in Figure 7.5.

A total of $11,000$ traces were generated for each arrhythmia type: $1,000$ negative samples (unsuccessful runs), and $1,000$ positive sample (successful runs) each run consisting of 1,000 time steps. Due to the nature of complete AV heart block, traces of good pacemaker operation would have been virtually identical and therefore, no good traces were created. Overall there was a total of $5,000$ positive examples, and $6,000$ negative examples (See table 7.5).

Traditionally, neural network classifiers for sequential data have been built around RNN architectures. With the recent advancements in transformer based classifiers, classifiers based on both transformers and LSTMs were trained in order to compare the difference in learning capability.

During training of both classifiers, each trace was broken into segments of $n \in \{20, 30, 50, 100\}$ lengths and the classifier was required to decide if the final action in the segment was correct or



(a) LSTM F1 scores across segment sizes.

(b) Transformer F1 scores across segment sizes.

Figure 7.6: $F1$ score box plots across different segment sizes for the 2 different architectures.

not. Training success was measured using the F1 [117] score, a common metric for comparing AI algorithms, and is presented in Table 7.6 where the P and R columns are the precision and recall scores. Figures 7.6a and 7.6b show the F1 scores for the LSTM and transformer respectively over different segment lengths. The LSTM significantly outperforms the transformer model.

Table 7.5: Positive and negative samples used for training rewards machine per arrhythmia type.

Arrhythmia Type	**# Positive Samples**	**# Negative Samples**
Complete AV block	—	1000
PVC	1000	1000
Mobitz II	1000	1000
Stochastic	1000	1000
Periodic Sinus Arrest	1000	1000
Healthy Heart	1000	1000
Total	5000	6000

Table 7.6: Reward Machine Training Results

Window Size	**LSTM**			**Transformer**		
	P	**R**	**F1**	**P**	**R**	**F1**
20	0.95 (0.003)	0.95 (0.004)	0.95 (0.004)	0.86 (0.007)	0.85 (0.007)	0.85 (0.07)
30	0.96 (0.007)	0.96 (0.007)	0.96 (0.007)	0.84 (0.01)	0.83 (0.01)	0.83 (0.01)
50	0.96 (0.007)	0.96 (0.007)	0.96 (0.007)	0.83 (0.02)	0.83 (0.02)	0.83 (0.02)
100	0.96 (0.005)	0.96 (0.006)	0.96 (0.006)	0.80 (0.01)	0.78 (0.02)	0.78 (0.02)

Algorithm 12 Pacemaker Agent Learning From Trace Log

Input: Episode Cnt (E), Replay Size (R), Log Len (L), Explore Rate (γ)
Output: Output: Trained RL Agent

```
 1: procedure FIT(E, L, γ)
 2:     for each Episode do
 3:         State ← S_0
 4:         Replay ← NewList()
 5:         for each Replay do
 6:             log ← NewList()
 7:             for each step in Log do
 8:                 Action ← NextAction(State, γ)
 9:                 State, Event ← Step(Action)
10:                 log.AddItem([Action, Event])
11:             end for
12:             Done ← Grade(log)
13:             Replay.AddItem(log, Done)
14:             if Done then
15:                 State ← S_0
16:             end if
17:         end for
18:         Train(Replay)
19:     end for
20: end procedure
```

7.3.2.2 Rewards to Pacemaker

Stochastic policy gradient RL (Section 4.2.1.2) was used for learning the pacemaker functionality. The basic algorithm used for this case study can be found in Algorithm 12. The reward machine does not work on individual actions but looks at a group of actions together and returns a normalized reward based only on correctness of the final action in the group. Lines 7–11 generate a log of pacemaker actions and intrinsic heart events at each step. When the log is full, it is graded by the rewards machine in line 12 and the log is added to the replay memory. If the rewards machine returned a low reward meaning actions in the log were incorrect, the pacemaker automaton is reset and the simulation restarted in lines 14–15. When the replay memory is full, the episode is over and the pacemaker agent is trained using the replay memory. The replay memory is then cleared and a new episode begins.

There are multiple parameters that control how the pacemaker agent learns. The exploration

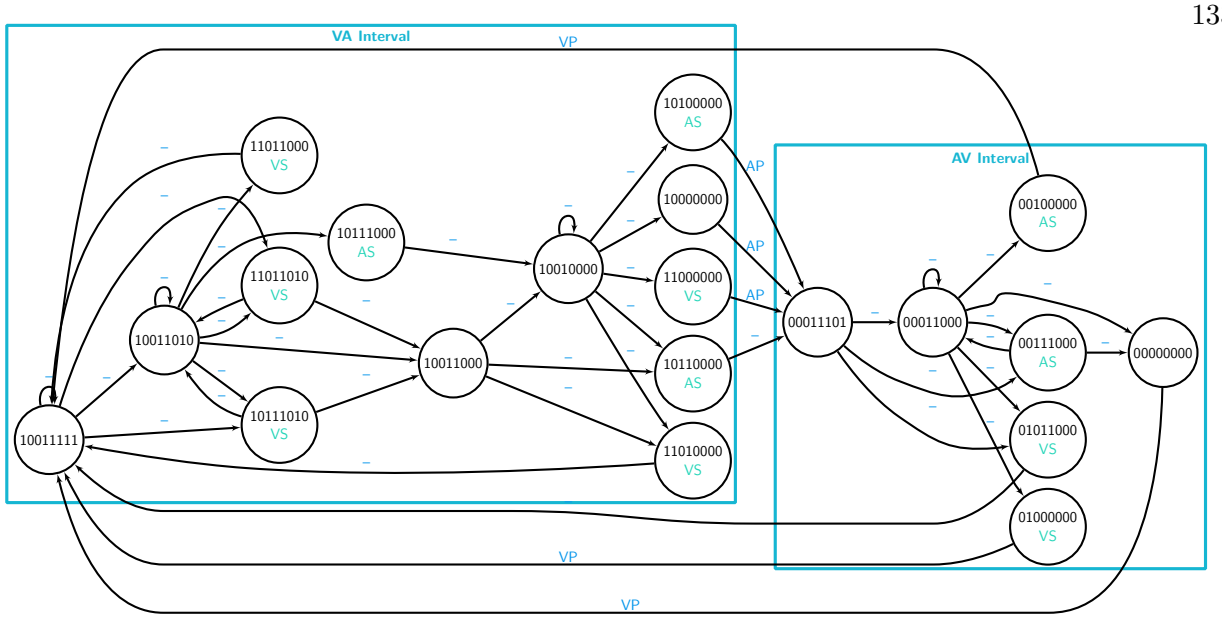


Figure 7.7: Learned State Machine — AS = Atrial Sense, VS = Ventricular Sense, AP = Atrial Pace, VP = Ventricular Pace

rate (γ) specifies how often a random action is selected. The log length defines how many steps are taken before grading the selected actions. This length must match the segment size used to train the rewards machine. The number of logs to be combined before training is defined by the replay memory size and episodes specifies the duration of training.

Training was performed a total of eight times, four each for each type of reward machine (LSTM and transformer) varying the segment or window sizes used as shown in Table 7.6. For each pacing agent training, the reward machines were trained with 20 folds and the pacemaker agent's log depth was set equal to the reward machines window size. The replay memory consisted of 200 logs and training ran for 50,000 episodes. The heart model was set to stochastic mode to assure the pacemaker agent was exposed to the widest possible variations in heart actions.

Validation was done using two methods. The first used the trained agent in prediction mode with the heart in each disease state over 2M steps representing approximately 56 days of continuous operation with a stochastic heart. No erroneous or omitted paces were observed. The second validation was through extraction of the learned automaton by recording each unique state transition seen and converting the resulting data into an automaton. Each agent returned the same

Table 7.7: Decoding Pacemaker State Name

Bit	Description
b7	Interval
b6	Ventricular Sense
b5	Atrial Sense
b4	AV or VA Interval Running
b3	Atrial Refractory (AR) Period
b2	Atrial Blanking (AB) Period
b1	Ventricular Refractory (VR) Period
b0	Ventricular Blanking (VB) Period

DFA (Figure 7.7) and, by visual inspection, found to be correct. In the figure, each state name is an encoding of the currently active periods, intervals, and intrinsic heart beats as described in Table 7.7. The graph edges represent the pacemaker agent's action (AP = Atrial Pace, VP = Ventricular Pace, -- = Wait) and there is only one valid action that can be taken in each state. There are two state transitions that may look incorrect. State 11000000, where a VS had occurred, is exited by an atrial pace. Likewise, state 00100000 exits with a VP even though an AS had occurred. This is acceptable behavior for a pacemaker. When an AV or VA interval ends, the pacemaker will pace regardless of heart activity. Doing otherwise would delay the required pacing pulse and violate the specification by pacing the patient below LRI. A check could be added before pace delivery but the intrinsic event could still fall between the check and the pace; there's no practical way to fully mitigate this.

7.4 Conclusions

Reward machines are difficult to design due to many factors. This chapter demonstrated various methods for reward machine synthesis that can result in a verifiably correct solution. In one method the reward machine is created as an automaton from a formal specification, as described in Chapter 6, that can be verified correct using standard formal verification. Automation of the process from formal specification to automaton also eliminates coding errors being injected during a manual reward machine creation.

The grid world case study showed that a reward machine automaton created from a formal specification can be used for training an RL agent to find an optimal policy in a stochastic environment. When the rules of the environment were changed to allow the agent to pause without restriction, it used that to its advantage to prevent falling into a trap highlighting a preference to prevent failure at the cost of never completing the task of reaching the goal. This shows that capturing and encouraging long-term goals in a specification is a difficult task.

The VVI pacemaker example showed that a reward machine can be used to train an RL agent to correctly control a cyclic stochastic system. Within the constraints of automaton creation process, the RL agent progressively learned to control systems with finer time resolution showing the ability to handle sparse rewards. The VVI pacemaker also showed the limits of Q-Learning as the state-space explosion inherent in the finer timed experiments became unsolvable. This limitation will be overcome with different RL architecture in a later chapter.

The second method of rewards machine synthesis was creation directly from traces generated from a black box source. By way of a case study, this was shown to be true. It was shown that two different sequential neural networks architectures can each successfully be trained solely from good and bad examples to execute correctly as a rewards machine. The quality of the rewards machine was verified through use in training an RL agent in emulating the original source of the traces.

The LSTM was far superior to the transformer in learning to correctly classify the pacemaker traces. Additionally, the transformer showed significantly more sensitivity to the length of the trace segments used for training with performance decreasing as length increased.

Training of the dual chamber pacemaker using policy gradient was significantly faster than the deep Q network used in for learning the VVI pacemaker even though the VVI pacemaker is simpler single chamber pacemaker. Another interesting secondary outcome came from collapsing the various period and interval timers to single bits for use as the state name as seen in Table 7.7 and Figure 7.7 allowed for extraction of the final learned DFA, which eased verification of the RL agent's pacemaker.

Chapter 8

Safe RL Training For Therapy Optimization Through Shielding

Chapter 6 presented a principled design method for specifying a real-time system based on Duration Calculus and showed the resulting formulas could be converted to an automaton and verified with standard formal methods tools to be correct. The automaton could then be deployed directly into the desired application. As such, there's limited utility in training an RL agent to duplicate the automaton's functionality. Furthermore, while the VVI pacemaker case study in Section 7.2.2 demonstrated that it possible for an RL agent to derive the functionality of a real-time system it also highlighted safety concerns when presented with operating conditions not represented in the training data. Even an almost surely satisfaction learning of the formal specification can still cause harm. Indeed, Bozkurt [23] and Venkataraman [150] show that RL can maximize the probability of satisfying stochastic system controllers with high degree and yet cannot guarantee reaching the optimal policy. Without that guarantee, RL optimized patient therapy is not possible.

While replacing a knowable controller with a trained RL agent is poor utility of RL's ability to explore and exploit unknown environments, used in conjunction with a known controller the combined strengths of both can create new and novel solutions. In this chapter, we will harness RL's strength by introducing permissive requirements that allow real-time exploration that is guaranteed safe through use of shielding (Section 4.2.1.3).

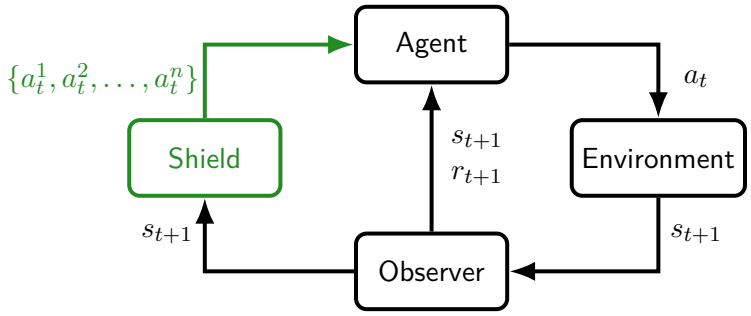


Figure 8.1: Shielded RL With Observer

8.1 Personalized Patient Therapy

Modern medical devices can generate significant data related to the patient's condition and disease state. Most medical devices, however, perform minimal analysis of this data in-situ and device therapy parameters are typically configured based on large population models for the average patient, leaving the individual patient with sub-optimal therapy. Optimizing each patient's therapy requires analysis of real-time in-situ patient data continuously adjusting patient therapy parameters that personalize the device to their specific needs. In-situ RL enables unique new therapies difficult to implement with traditional code to provide further improvement in patient quality of life.

Patient unique therapy requires the RL agent to continue training after device implant. If unconstrained, an RL pacemaker learning in-situ may take an action that can cause patient harm and, thus, a method is required to prevent inappropriate actions during learning. One solution to this is to add shielding that maintains safety when the RL agent attempts a hazardous action. With a shield in place, the RL agent is free to continue learning without risk of patient harm.

Shielded RL, as introduced in Section 4.2.1.3, prevents erroneous actions taken by the RL agent from reaching the environment where harm may occur. Figure 8.1 depicts the shielding model used in this chapter. It modifies the more generic shielding presented in Figure 4.6 by separating the environment into a model of the environment and an observer. This change enables the observer to be implemented as a rewards machine using the process shown in Chapter 7. With this configuration, the shield provides a set of allowable states based on the present environment

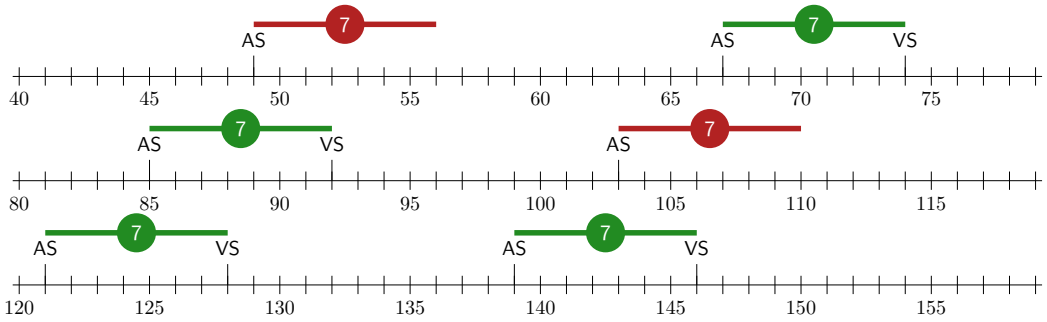
state. The agent sends the best action from the set based on past learning to the environment, which reacts and updates its state. The observer determines a reward based on how close to optimal the environment currently is, provides that to the agent and the cycle repeats.

8.2 Shielded RL For Real-Time Therapy Personalization

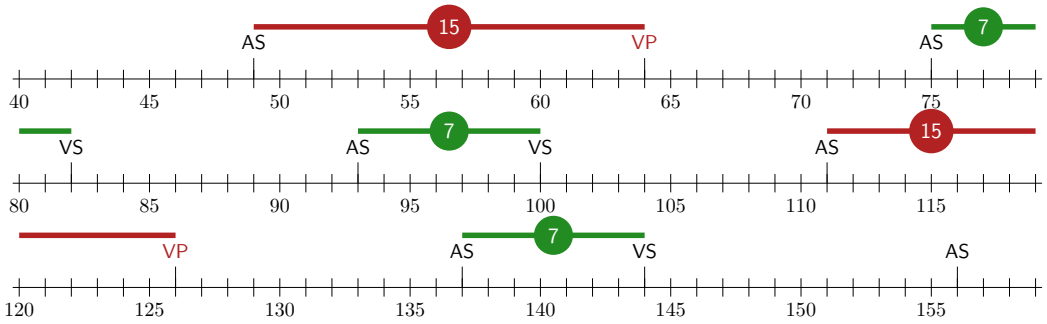
Mobitz II heart block is a condition where a ventricular beat is dropped due to AV node disfunction. The dropped beat has a periodicity such as every third beat. While current pacemaker therapy provides this missing beat, the timing of the beat is not optimal. Figure 8.2a depicts this where the AV interval is 7. The AV intervals with missing intrinsic ventricular beats are highlighted in red. Referring back to pacing therapy in Section 3.4, the pacemaker is preferential to intrinsic beats and will wait up until the lower rate expires before providing the needed beat. When the patient's current heart rate is similar to the lower rate, the beat will feel natural but if the patient has an elevated intrinsic rate, the pacemaker beat begins to feel dyssynchronous due to the difference in beat-to-beat timing of their elevated intrinsic rate and the slower pacing setting. Figure 8.2b depicts this irregularity where the intrinsic AV interval is 7 and the pacemaker lower rate AV interval is 15. Optimized therapy look similars to Figure 8.2c where the time between atrial beats and ventricular paces are close to the intrinsic time leading to a more natural feeling heart beat.

Solving this condition in procedural code would involve maintaining a running average of intrinsic beat timing and then matching that rate with the generated pace. Such a solution will have a time delay based on the depth of the running average and may continually lag the intrinsic rate as it changes in response to patient activity. RL, with its inherent ability to recognize patterns in data can detect changes much quicker to provide a more natural heart rate. This requires the RL to learn in-situ and must be constrained to still provide safe therapy during the learning process.

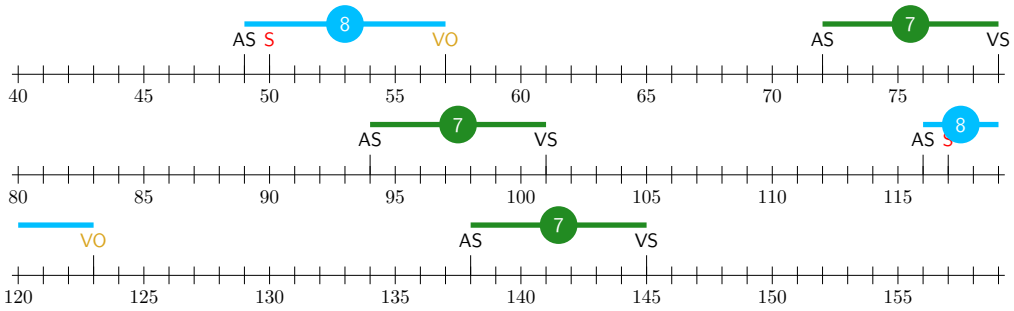
The first step to enabling this unique RL therapy is providing exploration space in the strict specification. Creating an exploration space requires a permissive specification that relaxes some rigid requirements. A pacemaker is required to pace at the end of the VA interval when no intrinsic beat is detected (DDD-13). Allowing the RL agent the option to pace anywhere after the completion



(a) Mobitz II — Untreated



(b) Mobitz II — Standard Pacing



(c) Mobitz II — Optimal Pacing

Figure 8.2: Mobitz II Pacing

of the ventricular refractory and the end of the AV interval creates exploration spece to find the optimal location for a ventricular beat based on the patient's intrinsic timing. Table 8.1 shows the DDD requirements that need modification for this case study. A new ventricular action, the permissive ventricular pace V_{other} (VO), is introduced.

Table 8.1: Permissive Pacemaker Case Study Specification Changes

Rule		Modified Specification
$permissivevp$	$=$	$\lceil VSN \Leftrightarrow Vother \rceil$
pav	$=$	$(AP \xhookrightarrow{Vother?}_{\texttt{pav}} PAV) \wedge (ASN \xhookrightarrow{Vother?}_{\texttt{sav}} SAV)$
		$\wedge((PAV \wedge \neg Vother) \curvearrowright EPAV) \wedge ((SAV \wedge \neg Vother) \curvearrowright ESAV)$
$vastart$	$=$	$(Vother \vee ESAV \vee EPAV \vee BOOT) \xhookrightarrow{ASN?}_{\texttt{va}} VA \wedge ((VA \wedge \neg ASN) \curvearrowright EVA)$

Unlike the VVI case study in Section 7.2.2 where the RL agent learned the entire pacemaker specification, here the RL is tasked with only identifying the optimal pacing time V_{other} and not the complete pacemaker. The standard pacemaker is controlling the heart as normal while the RL agent's actions preempt ventricular paces scheduled by the pacemaker. This leads to a modification of the shielded RL shown in Figure 8.1 adding the pacemaker as the primary environment controller. The complete design is shown in Figure 8.3 where the optimized therapy RL agent is separate from the primary pacemaker logic.

The reward machine for this therapy is a simple algorithm where the reward is proportional to how close the agent is to providing the missing beat at the optimal time, where optimal is exactly one clock tick later than when the intrinsic would have occurred — a nod to the desire to give the heart every possible chance to intrinsically beat. The permissible window for pacing is defined as a window of time starting at the length of the most recent SAV interval and ends with the Lower Rate Interval. The agent receives a reward of 5 for pacing at the optimal spot within this window. The reward decreases linearly by 1 for each tick later than optimal. A reward of zero is given if the pace is more than four beats late.

Shield design is straight forward. During the AV interval the shield provides the set of actions $A = \{pace, wait\}$ if a pace is allowed and would meets the upper and lower rate interval requirements. At all other times the set of actions is restricted to $A = \{wait\}$.

This will be demonstrated with a dual chamber pacemaker RL agent tailoring its pacing to match the heart's intrinsic rhythm and provide a more natural pacing feel to a patient with Mobitz

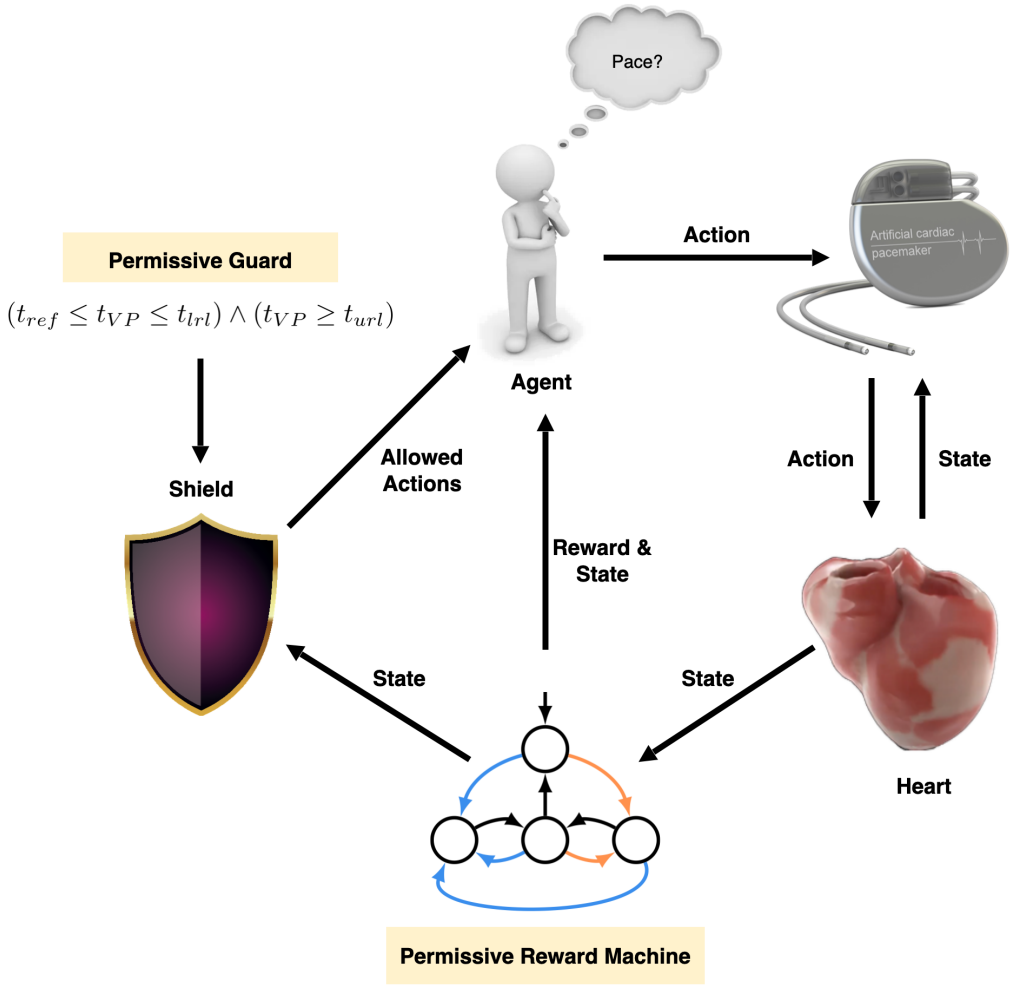
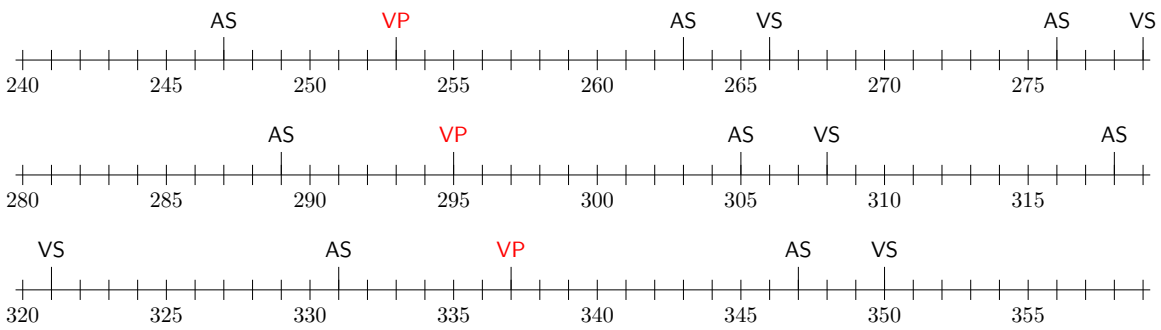


Figure 8.3: RL Agent Training With A Guard

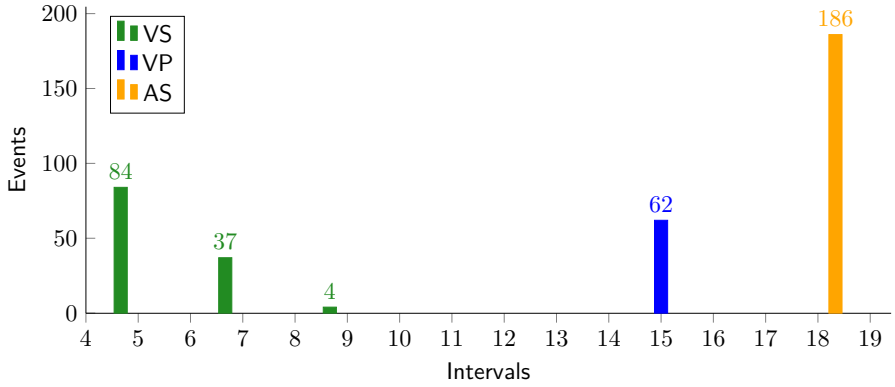
II heart block. This is a unique patient specific therapy not available in current pacemakers.

For the case study, the heart was allowed to randomly speed up and slow down to represent a patient engaging typical daily activities. As a control, the test set up was executed using the validated DDD automaton created from the strict specification in Chapter 6 to show RL agent learning standard pacing therapy. These results for Mobitz II arrhythmia are shown in Figure 8.4a. Note that intrinsic paces occur 3 time periods after an atrial intrinsic while the ventricular paces occur 6 time periods later. Figure 8.4b shows when each event occurred in relation to the cardiac cycle. The change in intrinsic heart rate can be seen with VS occurring at interval times of 5, 7, and 9. The pace maker generated a VP at a fixed interval of 15.

Figure 8.5a plots RL optimized pacing. The intrinsic AV interval has been extended to



(a) Pacing Timeline



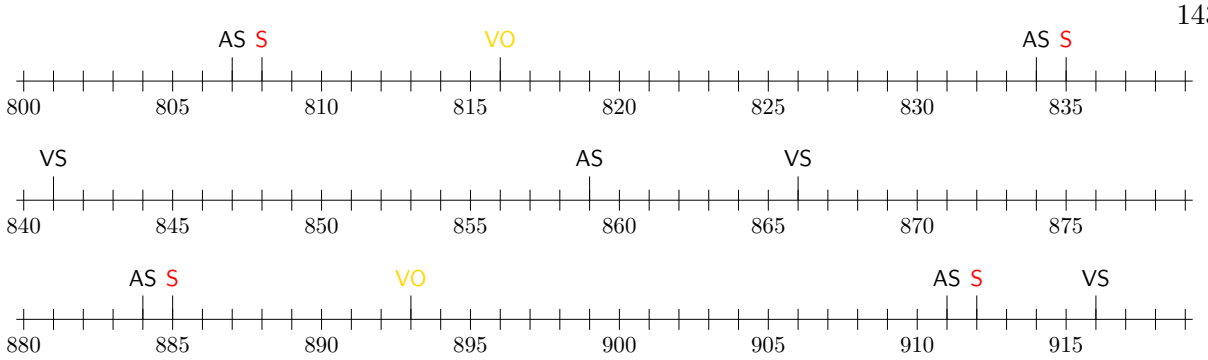
(b) Total Paces by Type

Figure 8.4: Standard Mobitz II Pacing

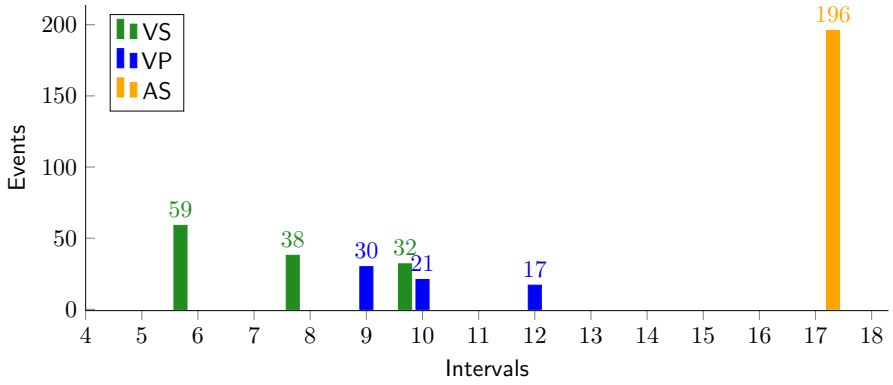
highlight the agent's improved therapy. In the figure, each S represents an incorrect VP action by the agent that has been corrected by the shield. The agent optimized paces are labeled as VO. The improvement is more visible in Figure 8.5b. In this plot, it can be seen that the VP now are within 3 time points of the VS. For example, when intrinsics were occurring at time interval 6, the associated VO occurred at interval 9. The association can be made through the ratio of VS at $t = 6$ (59) and VO at $t = 9$ (30) is indeed 3:2 as expected. The other VS-VO ratios are likewise the same with a small deviation to account for a beat delay for recognition of a rate change.

8.3 Conclusions

The power of RL is its explorative and exploitive behavior. Allowing a medical device RL agent to continually learn is-situ allows it to adapt to changing patient disease state and maintain



(a) Pacing Timeline



(b) Total Paces by Type

Figure 8.5: Optimal Mobitz II Pacing

optimal therapy. Doing so, however, brings significant patient risk if the continual learning process was allowed to try unsafe actions. Constraining the RL exploration through boundaries such as time periods when actions are acceptable and ranges of valid parameter values allows the agent to safely explore new policies without risk of harm.

Through the use of the Mobitz II case study, this chapter showed how shielded RL can effectively optimize a simple therapy without patient risk. Doing so required the introduction of a new concept in the permissive specification where strict requirements are relaxed to create exploration space for the RL agent. Creating this space becomes a new design challenge where the designer must understand the scope of the permutations created through this relaxation to assure there is no combination that can result in increased patient risk.

Chapter 9

Conclusions

In 2024, cardiology devices are project to approach US$73.42bn of an overall US$508bn market for medical devices of any kind with an annual growth rate of 5.58% over the period 2024-2029 [133]. This rapid market growth is putting pressure on device manufacturers to introduce newer, more complex devices into the market at a continually faster rate leading to shorter development times, leaving full testing and verification as a luxury [75].

AI can have a significant role to play in new medical device design that addresses both speed to market with new and novel therapies while providing greater safety assurances even with a decrease in testing. AI has the ability to improve capture of specifications from both the traditional method of natural language descriptions and directly from expert demonstrations into a formal specification that can be tested and validated to be correct. Taking advantage of reinforcement learning's exploration and exploitation abilities new and novel therapies can be designed quicker and if shielded, can continue to learn in-situ for real-time optimization of patient therapy.

Chapter 5 began with a novel new application of LLMs in converting analysis of surgical video into formal statements that describe the exact steps of a surgical process enabling a formal specification to be derived directly from expert demonstration. Such knowledge opens up new avenues for improved patient outcomes through grading of individual surgical technique and correlation of surgical procedure with patient outcome through fusion of vital signs with temporal surgical process steps. In addition, this descriptive technique of process to formulas allows creation of process models useable for off-line experimentation in process improvement without patient risk.

Formal specifications can describe desired behavior in succinct and unambiguous statements that are testable and can be proven correct. Many formal languages, however, are not sufficiently expressive to describe complex timing relationships while more expressive languages are neither decidable or deterministic. Chapter 6 introduced constrained versions of Duration Calculus, (EDC and TDC) that maintain the language's expressiveness while restoring both decidability and determinism. With these constrained dialects, formal specifications for pacemakers were presented in Tables 6.1 and 6.2.

Using RL for medical therapy requires a rewards machine that correctly enhances desired behavior. This is a difficult task to get correct. Creation from a natural language specification has been shown to be difficult and error prone. Chapter 7 presented two methods, one novel, for creating a rewards machine. A rewards machine can be an automaton which correlates current environment state with the desired optimal condition, generating a reward based on how close to optimal the environment is. When expressed as an automaton the rewards machine can be tested and validated before use to assure achieving the desired training outcome. The constrained DC dialects from Chapter 6 transition easily to automata allowing for using the formal pacemaker description to be directly used as a rewards machine that is provably correct. Knowing the design is correct decreases the level of testing and rework found in the standard design process.

A novel approach to rewards machine creation is using a neural network trained from traces exhibiting desired and undesired behavior. Such an approach enables rewards machine creation of an unknown black box through observation of excitations and responses. This is the first known application of a neural network as a rewards machine.

An unconstrained RL therapy is not without risk of patient harm. A trained RL agent can select incorrect actions when given situations not contained in the original training, while the exploration characteristic of RL while training will select unsafe actions if allowed to train in-situ. Preventing incorrect actions from reaching the environment enables real-time in-situ training for therapy optimization and creation of novel new therapies. Chapter 8 introduced shielded RL where the RL agent was only supplied with valid actions that it could select from. Control algorithms

in design specifications are typically rigid, without room for exploration. This chapter introduced the novel concept of a permissive requirement that defined allowable parameter permutations that maintained safety while allowing for optimization. This was demonstrated with a real-world demonstration for optimization of pacing therapy.

While this thesis used a pacemaker as an example for using RL safely to create therapies optimized to the individual patient, the techniques are not pacemaker unique. What has been presented here is applicable to any medical device. Using these techniques cannot only address the desire for shorter design cycles, it enables new device manufacturer unique therapies optimized directly for the patient that are guaranteed safe.

This thesis began with textual descriptions and specifications of a life sustaining medical device. With each succeeding chapter these were translated into reward mechanisms capable of training RL agents to devise new, novel, and safe therapies uniquely optimized for each patient.

Truly,

From text to life.

Bibliography

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

[2] Mohammad Afzal, Sankalp Gambhir, Ashutosh Gupta, S. Krishna, Ashutosh Trivedi, and Alvaro Velasquez. Ltl-based non-markovian inverse reinforcement learning. In Noa Agmon, Bo An, Alessandro Ricci, and William Yeoh, editors, Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023, London, United Kingdom, 29 May 2023 - 2 June 2023, pages 2857–2859. ACM, 2023.

[3] Lisa NF Aird, Dennis Hong, Scott Gmora, Ruth Breau, and Mehran Anvari. The impact of a standardized program on short and long-term outcomes in bariatric surgery. Surgical endoscopy, 31:801–808, 2017.

[4] Masood Akhtar, Mohammad R Jazayeri, Jasbir Sra, Zalmen Blanck, Sanjay Deshpande, and Anwer Dhala. Atrioventricular nodal reentry. clinical, electrophysiological, and therapeutic considerations. Circulation, 88(1):282–295, 1993.

[5] Hassan Al Hajj, Mathieu Lamard, Pierre-Henri Conze, Soumali Roychowdhury, Xiaowei Hu, Gabija Maršalkaitė, Odysseas Zisimopoulos, Muneer Ahmad Dedmari, Fenqiang Zhao, Jonas Prellberg, Manish Sahu, Adrian Galdran, Teresa Araújo, Duc My Vo, Chandan Panda, Navdeep Dahiya, Satoshi Kondo, Zhengbing Bian, Arash Vahdat, Jonas Bialopetravičius, Evangello Flouty, Chenhui Qiu, Sabrina Dill, Anirban Mukhopadhyay, Pedro Costa, Guilherme Aresta, Senthil Ramamurthy, Sang-Woong Lee, Aurélio Campilho, Stefan Zachow, Shunren Xia, Sailesh Conjeti, Danail Stoyanov, Jogundas Armaitis, Pheng-Ann Heng, William G. Macready, Béatrice Cochener, and Gwenolé Quellec. Cataracts: Challenge on automatic tool annotation for cataract surgery. Medical Image Analysis, 52:24–41, 2019.

[6] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In Proceedings of the AAAI conference on artificial intelligence, volume 32, 2018.

[7] Rajeev Alur, Osbert Bastani, Kishor Jothimurugan, Mateo Perez, Fabio Somenzi, and Ashutosh Trivedi. Policy synthesis and reinforcement learning for discounted ltl. arXiv preprint arXiv:2305.17115, 2023.

[8] Rajeev Alur and David L Dill. A theory of timed automata. Theoretical computer science, 126(2):183–235, 1994.

[9] Rajeev Alur, Limor Fix, and Thomas A Henzinger. Event-clock automata: A determinizable class of timed automata. Theoretical Computer Science, 211(1-2):253–273, 1999.

[10] Ann Arbor Electrogram Libraries. `http://www.electrogram.com`.

[11] Charles Antzelevitch and Alexander Burashnikov. Mechanisms of cardiac arrhythmia. Electrical Diseases of the Heart: Volume 1: Basic Foundations and Primary Electrical Diseases, pages 93–128, 2013.

[12] T Armstrong, D Yu, A Frischknecht, R Minter, P Andreatta, and S Kasten. Standardization of surgical procedures for identifying best practices and training. Work, 41(Supplement 1):4673–4679, 2012.

[13] Krzysztof Badura, Dominika Buławska, Bartłomiej Dąbek, Alicja Witkowska, Wiktoria Lisińska, Ewa Radzioch, Sylwia Skwira, Ewelina Młynarska, Jacek Rysz, and Beata Franczyk. Primary electrical heart disease—principles of pathophysiology and genetics. International Journal of Molecular Sciences, 25(3):1826, 2024.

[14] Christel Baier and Joost-Pieter Katoen. Principles of model checking. MIT press, 2008.

[15] Earl Bakken. One Man's Full Life. Medtronic, Inc, 1999.

[16] Peter Belott and Dwight Reynolds. Permanent pacemaker and implantable cardioverter-defibrillator implantation in adults. In Clinical Cardiac Pacing, Defibrillation and Resynchronization Therapy, pages 631–691. Elsevier, 2017.

[17] Alan D Bernstein, Jean-Claude Daubert, Ross D Fletcher, David L Hayes, Berndt Lüderitz, Dwight W Reynolds, Mark H Schoenfeld, and Richard Sutton. The revised NASPE/BPEG generic code for antibradycardia, adaptive-rate, and multisite pacing. Pacing and clinical electrophysiology, 25(2):260–264, 2002.

[18] Sandra I Berrios-Torres et al. Centers for Disease Control and Prevention Guideline for the Prevention of Surgical Site Infection. Healthcare Infection Control Practices Advisory Committee (2017), 2017. https://doi.org/10.1001/jamasurg.2017.0904.

[19] Jean Bézivin and Olivier Gerbé. Towards a precise definition of the omg/mda framework. In Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001), pages 273–280. IEEE, 2001.

[20] Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. Shield synthesis: Runtime enforcement for reactive systems. In International conference on tools and algorithms for the construction and analysis of systems, pages 533–548. Springer, 2015.

[21] Silvia Bonfanti, Angelo Gargantini, and Atif Mashkoor. A systematic literature review of the use of formal methods in medical software systems. Journal of Software: Evolution and Process, 30(5):e1943, 2018.

[22] Esther M Bonrath, Lauren E Gordon, and Teodor P Grantcharov. Characterising 'near miss' events in complex laparoscopic surgery through video analysis. BMJ quality & safety, 24(8):516–521, 2015.

[23] Alper Kamil Bozkurt, Yu Wang, Michael M Zavlanos, and Miroslav Pajic. Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In 2020 IEEE International Conference on Robotics and Automation (ICRA), pages 10349–10355. IEEE, 2020.

[24] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. arXiv preprint arXiv:1606.01540, 2016.

[25] Peter H Brubaker and Dalane W Kitzman. Chronotropic incompetence: causes, consequences, and management. Circulation, 123(9):1010–1020, 2011.

[26] Sok-Sithikun Bun, Decebal Gabriel Latcu, Francis Marchlinski, and Nadir Saoudi. Atrial flutter: more than just one of a kind. European Heart Journal, 36(35):2356–2363, 04 2015.

[27] Paola Busia, Andrea Cossettini, Thorir Mar Ingolfsson, Simone Benatti, Alessio Burrello, Moritz Scherer, Matteo Antonio Scrugli, Paolo Meloni, and Luca Benini. Eegformer: Transformer-based epilepsy detection on raw eeg traces for low-channel-count wearable continuous monitoring devices. In 2022 IEEE Biomedical Circuits and Systems Conference (BioCAS), pages 640–644. IEEE, 2022.

[28] Franck Cassez and Kim Larsen. The impressive power of stopwatches. In Catuscia Palamidessi, editor, CONCUR 2000 — Concurrency Theory, pages 138–152, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[29] Zhou Chaochen, Michael R Hansen, and Peter Sestoft. Decidability and undecidability results for duration calculus. In Annual Symposium on Theoretical Aspects of Computer Science, pages 58–68. Springer, 1993.

[30] Zhou Chaochen, Charles Anthony Richard Hoare, and Anders P Ravn. A calculus of durations. Information processing letters, 40(5):269–276, 1991.

[31] Kai Jye Chee and Dzati Athiar Ramli. Electrocardiogram biometrics using transformer's self-attention mechanism for sequence pair feature extractor and flexible enrollment scope identification. Sensors, 22(9):3446, 2022.

[32] Zhe Chen, Hugh T Blair, and Jason Cong. Energy-efficient lstm inference accelerator for real-time causal prediction. ACM Transactions on Design Automation of Electronic Systems (TODAES), 27(5):1–19, 2022.

[33] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. Journal of Machine Learning Research, 24(240):1–113, 2023.

[34] Demetra D Christou and Douglas R Seals. Decreased maximal heart rate with aging is related to reduced β-adrenergic responsiveness but is largely explained by a reduction in intrinsic heart rate. Journal of applied physiology, 105(1):24–29, 2008.

[35] Jack Clark and Dario Amodei. Faulty reward functions in the wild. Internet: https://blog. openai. com/faulty-reward-functions, 2016. Accessed: 2024-11-12.

[36] Edmund M Clarke and E Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In Workshop on logic of programs, pages 52–71. Springer, 1981.

[37] Francisco G Cosío. Atrial flutter, typical and atypical: a review. Arrhythmia & electrophysiology review, 6(2):55, 2017.

[38] David Da Costa, William J Brady, and June Edhouse. Bradycardias and atrioventricular conduction block. Bmj, 324(7336):535–538, 2002.

[39] Giuseppe De Giacomo and Moshe Vardi. Synthesis for ltl and ldl on finite traces. In Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015.

[40] Hugo Cardoso de Souza, Victor Ribeiro Marques, Mateus Fernandes da Silva, and Juliana de Castro Solano. Bundle branch block: right and left prognosis implications. ARCHIVOS DE MEDICINA, 2(1):7, 2016.

[41] Justin B Dimick and Oliver A Varban. Surgical video analysis: an emerging tool for improving surgeon performance, 2015.

[42] Council directive 2017/745. on medical devices, amending directive 2001/83/EC, Regulation (EC) No 178/2002 and Regulation (EC) No 1223/2009 and repealing Council Directives 90/385/EEC and 93/42/EEC. Official Journal L117/1, April 2017.

[43] Kalyani Dole, Ashutosh Gupta, John Komp, Shankaranarayanan Krishna, and Ashutosh Trivedi. Event-triggered and time-triggered duration calculus for model-free reinforcement learning. In 2021 IEEE Real-Time Systems Symposium (RTSS), pages 240–252. IEEE, 2021.

[44] Kalyani Dole, Ashutosh Gupta, John Komp, Shankaranarayanan Krishna, and Ashutosh Trivedi. Correct-by-construction reinforcement learning of cardiac pacemakers from duration calculus requirements. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 37, pages 14792–14800, 2023.

[45] David Duncker and Christian Veltmann. Optimizing antitachycardia pacing: back to the roots, 2017.

[46] Ann-marie Ericsson, Robert Nilsson, and Sten F Andler. Operator patterns for analysis of composite events in timed automata. In WIP Proceedings: 24th IEEE Real-Time Systems Symposium, Cancun, Mexico, pages 155–159, 2003.

[47] Tom Everitt, Victoria Krakovna, Laurent Orseau, Marcus Hutter, and Shane Legg. Reinforcement learning with a corrupted reward channel. arXiv preprint arXiv:1705.08417, 2017.

[48] Goerschwin Fey, Tara Ghasempouri, Swen Jacobs, Gianluca Martino, Jaan Raik, and Heinz Riener. Design understanding: From logic to specification. In 2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), pages 172–175. IEEE, 2018.

[49] Gregory A. Fishbein, Michael C. Fishbein, Jessica Wang, and L. Maximilian Buja. Chapter 10 - myocardial ischemia and its complications. In L. Maximilian Buja and Jagdish Butany, editors, Cardiovascular Pathology (Fifth Edition), pages 407–445. Academic Press, fifth edition edition, 2022.

[50] U.S. Food and Drug Administration (FDA). General principles of software validation; final guidance for industry and FDA staff, version 2.0. `https://www.fda.gov/media/73141/download`, Jan 2002. Accessed: 2020-11-1.

[51] U.S. Food and Drug Administration (FDA). White paper: infusion pump improvement initiative. Center for Devices and Radiological Health, 2010.

[52] US Food and Drug Administration (FDA). Medical device recall report FY2003 to FY2012. Center for Devices and Radiological Health, 2012.

[53] U.S. Food and Drug Administration (FDA). Food and drugs, 21 C.F.R., § 820. Code of Federal Regulations, Government Printing Office, 2020.

[54] Kevin Forsberg and Harold Mooz. The relationship of system engineering to the project cycle. Center for Systems Management, 5333, 1991.

[55] Ian Fox, Joyce Lee, Rodica Pop-Busui, and Jenna Wiens. Deep reinforcement learning for closed-loop blood glucose control. In Machine Learning for Healthcare Conference, pages 508–536. PMLR, 2020.

[56] L.A. Geddes and Earl E. Bakken. Retroscope - who first performed cardiac pacing: why, when, and where? IEEE Engineering in Medicine and Biology Magazine, 26(3):77–79, 2007.

[57] Gilles Geeraerts, Jean-François Raskin, and Nathalie Sznajder. Event clock automata: From theory to practice. In International Conference on Formal Modeling and Analysis of Timed Systems, pages 209–224. Springer, 2011.

[58] Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. arXiv preprint arXiv:2012.14913, 2020.

[59] Anne M Gillis. Pacing for sinus node disease. In Clinical Cardiac Pacing, Defibrillation and Resynchronization Therapy, pages 375–398. Elsevier, 2017.

[60] Artur Oliveira Gomes and Marcel Vinicius Medeiros Oliveira. Formal specification of a cardiac pacing system. In International Symposium on Formal Methods, pages 692–707. Springer, 2009.

[61] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. Neural networks, 18(5-6):602–610, 2005.

[62] Ben Greenman, Sam Saarinen, Tim Nelson, and Shriram Krishnamurthi. Little tricky logic: misconceptions in the understanding of ltl. arXiv preprint arXiv:2211.01677, 2022.

[63] Ashutosh Gupta, John Komp, Abhay Singh Rajput, Krishna Shankaranarayanan, Ashutosh Trivedi, and Namrita Varshney. Integrating explanations in learning ltl specifications from demonstrations. arXiv preprint arXiv:2404.02872, 2024.

[64] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. arXiv:1801.01290 [cs, stat], January 2018.

[65] Christopher Hahn, Frederik Schmitt, Julia J Tillman, Niklas Metzger, Julian Siber, and Bernd Finkbeiner. Formal specifications from natural language. arXiv preprint arXiv:2206.01962, 2022.

[66] Matthew Kendall Hawks, Madison LB Paul, and Omojo Odihi Malu. Sinus node dysfunction. American family physician, 104(2):179–185, 2021.

[67] Jesper G Henriksen, Jakob Jensen, Michael Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders Sandholm. Mona: Monadic second-order logic in practice. In Tools and Algorithms for the Construction and Analysis of Systems: First International Workshop, TACAS'95 Aarhus, Denmark, May 19–20, 1995 Selected Papers 1, pages 89–110. Springer, 1995.

[68] Thomas A Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In Automata, Languages and Programming: 19th International Colloquium Wien, Austria, July 13–17, 1992 Proceedings 19, pages 545–558. Springer, 1992.

[69] S Hochreiter. Long short-term memory. Neural Computation MIT-Press, 1997.

[70] Melissa E Hogg, Mazen Zenati, Stephanie Novak, Yong Chen, Yan Jun, Jennifer Steve, Stacy J Kowalsky, David L Bartlett, Amer H Zureikat, and Herbert J Zeh III. Grading of surgeon technical performance predicts postoperative pancreatic fistula for pancreaticoduodenectomy independent of patient-related variables. Annals of surgery, 264(3):482–491, 2016.

[71] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. arXiv preprint arXiv:2311.05232, 2023.

[72] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. Journal of Artificial Intelligence Research, 73:173–208, 2022.

[73] Paul N Jensen, Noelle N Gronroos, Lin Y Chen, Aaron R Folsom, Chris Defilippi, Susan R Heckbert, and Alvaro Alonso. Incidence of and risk factors for sick sinus syndrome in the general population. Journal of the American College of Cardiology, 64(6):531–538, 2014.

[74] Zhihao Jiang, Miroslav Pajic, Salar Moarref, Rajeev Alur, and Rahul Mangharam. Modeling and verification of a dual chamber implantable pacemaker. In International conference on tools and algorithms for the construction and analysis of systems, pages 188–203. Springer, 2012.

[75] Erez Kaminski. Why the epidemic in quality failures and what medtechs can learn. Forbes, 2024.

[76] Jason Kaplan, Arjun Kanwal, Intisar Ahmed, and Vasimahmed Lala. Reentrant arrhythmias. Europe PMC, 2023.

[77] G Neal Kay and Richard B Shepard. Stimulation and excitation of cardiac tissues. In Kenneth A Ellenbogen, Bruce L Wilkoff, G Neal Kay, and Chu Pak Lau, editors, Clinical Cardiac Pacing, Defibrillation and Resynchronization Therapy, pages 61–113. Elsevier, 2017.

[78] Timothy Patrick Kelly. Arguing safety: a systematic approach to managing safety cases. PhD thesis, University of York York, UK, 1999.

[79] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In 2019 international conference on robotics and automation (ICRA), pages 8248–8254. IEEE, 2019.

[80] Ralf Kneuper. Die geschichtliche entwicklung des v-modells. Technical report, IUBH Discussion Papers-IT & Engineering, 2018.

[81] John Komp, Dananjay Srinivas, Maria Pacheco, and Ashutosh Trivedi. Show, don't tell: Learning reward machines from demonstrations for reinforcement learning-based cardiac pacemaker synthesis. arXiv preprint arXiv:2411.01750, 2024.

[82] Savas Konur. A survey on temporal logics for specifying and verifying real-time systems. Frontiers of Computer Science, 7:370–403, 2013.

[83] Hermann Kopetz. Time-triggered real-time computing. IFAC Proceedings Volumes, 35(1):59–70, 2002.

[84] Ron Koymans. Specifying real-time properties with metric temporal logic. Real-time systems, 2(4):255–299, 1990.

[85] Orna Kupferman and MosheY Vardi. Model checking of safety properties. In International Conference on Computer Aided Verification, pages 172–183. Springer, 1999.

[86] Software Quality Research Laboratory. Pacemaker system specification. `http://sqrl.mcmaster.ca/_SQRLDocuments/PACEMAKER.pdf`, 2007. Accessed: 2020-11-1.

[87] Paras Lakhani, Adam B Prater, R Kent Hutson, Kathy P Andriole, Keith J Dreyer, Jose Morey, Luciano M Prevedello, Toshi J Clark, J Raymond Geis, Jason N Itri, et al. Machine learning in radiology: applications beyond image interpretation. Journal of the American College of Radiology, 15(2):350–359, 2018.

[88] Axel van Lamsweerde. Formal specification: a roadmap. In Proceedings of the Conference on the Future of Software Engineering, pages 147–159, 2000.

[89] Brian R Larson. Formal semantics for the pacemaker system specification. ACM SIGAda Ada Letters, 34(3):47–60, 2014.

[90] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. General ltl specification mining (t). In 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 81–92. IEEE, 2015.

[91] Nancy Leveson et al. Medical devices: The therac-25. Appendix of: Safeware: System Safety and Computers, 1995.

[92] David E Lightfoot. Formal specification using Z. Macmillan International Higher Education, 1991.

[93] Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, and Li Zhang. Exploring and evaluating hallucinations in llm-powered code generation. arXiv preprint arXiv:2404.00971, 2024.

[94] Hanchao Liu, Wenyuan Xue, Yifei Chen, Dapeng Chen, Xiutian Zhao, Ke Wang, Liping Hou, Rongjun Li, and Wei Peng. A survey on hallucination in large vision-language models. arXiv preprint arXiv:2402.00253, 2024.

[95] Constantinos Loukas. Video content analysis of surgical procedures. Surgical endoscopy, 32:553–568, 2018.

[96] Yantian Luo, Xu Chen, Ning Ge, Wei Feng, and Jianhua Lu. Transformer-based malicious traffic detection for internet of things. In ICC 2022-IEEE International Conference on Communications, pages 4187–4192. IEEE, 2022.

[97] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In International symposium on formal techniques in real-time and fault-tolerant systems, pages 152–166. Springer, 2004.

[98] J Michael Mangrum and John P DiMarco. The evaluation and management of bradycardia. New England Journal of Medicine, 342(10):703–709, 2000.

[99] Gregory M Marcus. Evaluation and management of premature ventricular complexes. Circulation, 141(17):1404–1418, 2020.

[100] Seth S Martin, Aaron W Aday, Zaid I Almarzooq, Cheryl AM Anderson, Pankaj Arora, Christy L Avery, Carissa M Baker-Smith, Bethany Barone Gibbs, Andrea Z Beaton, Amelia K Boehme, et al. 2024 heart disease and stroke statistics: A report of us and global data from the american heart association. Circulation, 149(8):e711–e740, 2024.

[101] John H McAnulty and Shahbudin H Rahimtoola. Bundle branch block. Progress in cardiovascular diseases, 26(4):333–354, 1984.

[102] Medtronic, Inc. ADVISA™ DR A4DR01.

[103] Medtronic, Inc. 2290 Analyzer Reference Guide, 2012.

[104] Dominique Méry, Bernhard Schätz, and Alan Wassyng. The pacemaker challenge: developing certifiable medical devices (dagstuhl seminar 14062). In Dagstuhl Reports, volume 4.2. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[105] Dominique Méry and Neeraj Kumar Singh. Formal specification of medical systems by proof-based refinement. ACM Transactions on Embedded Computing Systems (TECS), 12(1):1–25, 2013.

[106] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. Chip placement with deep reinforcement learning. arXiv preprint arXiv:2004.10746, 2020.

[107] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. arXiv preprint arXiv:2410.05229, 2024.

[108] Mungojerrie–formal reinforcement learning. accessed: 05/28/2021.

[109] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. nature, 518(7540):529–533, 2015.

[110] Mustafa Abshir Mohamed, Geylani Kardas, and Moharram Challenger. A systematic literature review on model-driven engineering for cyber-physical systems. arXiv preprint arXiv:2103.08644, 2021.

[111] Shahmir Khan Mohammed, Shakti Singh, Rabeb Mizouni, and Hadi Otrok. Trace: Transformer-based continuous tracking framework using iot and mcs. Journal of Network and Computer Applications, 222:103793, 2024.

[112] John Morellato, William Chik, MA Barry, Juntang Lu, Aravinda Thiagalingam, Pramesh Kovoor, and Jim Pouliopoulos. Quantitative spectral assessment of intracardiac electrogram characteristics associated with post infarct fibrosis and ventricular tachycardia. PLoS One, 13(10):e0204997, 2018.

[113] Anitha Murugesan, Mats PE Heimdahl, Michael W Whalen, Sanjai Rayadurgam, John Komp, Lian Duan, Baek-Gyu Kim, Oleg Sokolsky, and Insup Lee. From requirements to code: Model based development of a medical cyber physical system. In Software Engineering in Health Care: 4th International Symposium, FHIES 2014, and 6th International Workshop, SEHC 2014, Washington, DC, USA, July 17-18, 2014, Revised Selected Papers 4, pages 96–112. Springer, 2017.

[114] P. K. Pandya. Specifying and deciding quantified discrete-time duration calculus formulae using dcvalid: An automata theoretic approach. In Proceedings of RTTOOLS, 2001.

[115] Eike Petersen, Yannik Potdevin, Esfandiar Mohammadi, Stephan Zidowitz, Sabrina Breyer, Dirk Nowotka, Sandra Henn, Ludwig Pechmann, Martin Leucker, Philipp Rostalski, et al. Responsible and regulatory conform machine learning for medicine: a survey of challenges and solutions. IEEE Access, 10:58375–58418, 2022.

[116] Amir Pnueli. The temporal logic of programs. In 18th annual symposium on foundations of computer science (sfcs 1977), pages 46–57. ieee, 1977.

[117] David MW Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. arXiv preprint arXiv:2010.16061, 2020.

[118] Zhilin Qu, Michael B Liu, Riccardo Olcese, and James N Weiss. R-on-t and the initiation of reentry revisited: Integrating old and new concepts. Heart Rhythm, 1:15, 2022.

[119] J Richard. Büchi. on a decision method in restricted second-order arithmetic. In Proceedings of the International Congress on Logic, Math, and Philosophy of Science. Standford University Press, 1962.

[120] W. W. Royce. Managing the development of large software systems: concepts and techniques. In Proceedings IEEE WESTCON. IEEE Computer Society Press, 1970.

[121] George J Rozanski, Jose Jalifé, and Gordon K Moe. Reflected reentry in nonhomogeneous ventricular muscle as a mechanism of cardiac arrhythmias. Circulation, 69(1):163–173, 1984.

[122] Luis F Santana, Edward P Cheng, and W Jonathan Lederer. How does the shape of the cardiac action potential control calcium signaling and contraction in the heart? Journal of molecular and cellular cardiology, 49(6):901, 2010.

[123] Alfred Sarkissian. An exploratory analysis of US FDA Class I medical device recalls: 2014–2018. Journal of medical engineering & technology, 42(8):595–603, 2018.

[124] Peter Schachinger and Hans L Johannesson. Computer modelling of design specifications. Journal of engineering design, 11(4):317–329, 2000.

[125] Dieter Scheithauer and Kevin Forsberg. 4.5. 3 v-model views. In INCOSE International Symposium, volume 23, pages 502–516. Wiley Online Library, 2013.

[126] David Scherf. The mechanism of flutter and fibrillation. American Heart Journal, 71(2):273–280, 1966.

[127] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic Policy Gradient Algorithms. In ICML, June 2014.

[128] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. nature, 550(7676):354–359, 2017.

[129] Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. Advances in Neural Information Processing Systems, 35:9460–9471, 2022.

[130] British Heart Rythm Society. The ecg/egm challenge. `https://bhrs.com/ecg-egm-challenge/`, 2024. Accessed: 2024-05-19.

[131] David H Spodick. Normal sinus heart rate: appropriate rate thresholds for sinus tachycardia and bradycardia. Southern medical journal, 89(7):666–667, 1996.

[132] Jeremy Sproston. Discrete-time verification and control for probabilistic rectangular hybrid automata. In 2011 Eighth International Conference on Quantitative Evaluation of SysTems, pages 79–88. IEEE, 2011.

[133] statista. Medical devices - worldwide. 2024.

[134] John A Stauffer, Edwin O Onkendi, Michael B Wallace, Massimo Raimondo, Timothy A Woodward, Frank J Lukens, and Horacio J Asbun. Standardization and streamlining of a pancreas surgery practice improves outcomes and resource utilization: a single institution's 20-year experience. The American Journal of Surgery, 214(3):450–455, 2017.

[135] Gary L Stiles, Marc G Caron, and Robert J Lefkowitz. Beta-adrenergic receptors: biochemical mechanisms of physiological regulation. Physiological reviews, 64(2):661–743, 1984.

[136] Nils Strodthoff, Patrick Wagner, Tobias Schaeffter, and Wojciech Samek. Deep learning for ecg analysis: Benchmarks and insights from ptb-xl. IEEE journal of biomedical and health informatics, 25(5):1519–1528, 2020.

[137] P Vijay Suman, Paritosh K Pandya, Shankara Narayanan Krishna, and Lakshmi Manasa. Timed automata with integer resets: Language inclusion and expressiveness. In International conference on formal modeling and analysis of timed systems, pages 78–92. Springer, 2008.

[138] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. MIT press, 2018.

[139] Richard S Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In S. A. Solla, T. K. Leen, and K. Müller, editors, Advances in Neural Information Processing Systems 12, pages 1057–1063. MIT Press, 2000.

[140] Charles Swerdlow, Mark Brown, and Pierre Bordachar. Sensing and detection with cardiac implantable electronic devices. In Clinical Cardiac Pacing, Defibrillation and Resynchronization Therapy, pages 114–167. Elsevier, 2017.

[141] Elizabeth Thong, Ayesha Ahmed, and Kenneth T. MacLeod. An Introduction to the Cardiac Action Potentials, pages 49–59. Springer International Publishing, Cham, 2019.

[142] SM Tonmoy, SM Zaman, Vinija Jain, Anku Rani, Vipula Rawte, Aman Chadha, and Amitava Das. A comprehensive survey of hallucination mitigation techniques in large language models. arXiv preprint arXiv:2401.01313, 2024.

[143] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971, 2023.

[144] Hoang Duy Trinh, Lorenza Giupponi, and Paolo Dini. Urban anomaly detection by processing mobile traffic traces with lstm neural networks. In 2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), pages 1–8. IEEE, 2019.

[145] Gary Tse. Mechanisms of cardiac arrhythmias. Journal of arrhythmia, 32(2):75–81, 2016.

[146] Teun van de Laar, Zengjie Zhang, Shuhao Qi, Sofie Haesaert, and Zhiyong Sun. Vernacopter: Disambiguated natural-language-driven robot via formal specifications. arXiv preprint arXiv:2409.09536, 2024.

[147] Madhu Babu Vankadari, Kaushik Das, Chinmay Shinde, and Swagat Kumar. A reinforcement learning approach for autonomous control and landing of a quadrotor. In 2018 International Conference on Unmanned Aircraft Systems (ICUAS), pages 676–683. IEEE, 2018.

[148] Moshe Y Vardi and Pierre Wolper. Reasoning about infinite computations. Information and computation, 115(1):1–37, 1994.

[149] A Vaswani. Attention is all you need. Advances in Neural Information Processing Systems, 2017.

[150] Harish Venkataraman, Derya Aksaray, and Peter Seiler. Tractable reinforcement learning of signal temporal logic objectives. In Learning for Dynamics and Control, pages 308–317. PMLR, 2020.

[151] Pugazhendhi Vijayaraman and Kenneth A Ellenbogen. Atrioventricular conduction system disease. In Kenneth A Ellenbogen, Bruce L Wilkoff, G Neal Kay, and Chu Pak Lau, editors, Clinical Cardiac Pacing, Defibrillation and Resynchronization Therapy, pages 399–453. Elsevier, 2017.

[152] Dolores R Wallace and D Richard Kuhn. Failure modes in medical device software: an analysis of 15 years of recall data. International Journal of Reliability, Quality and Safety Engineering, 8(04):351–371, 2001.

[153] Tony Tong Wang, Adam Gleave, Tom Tseng, Kellin Pelrine, Nora Belrose, Joseph Miller, Michael D Dennis, Yawen Duan, Viktor Pogrebniak, Sergey Levine, et al. Adversarial policies beat superhuman go ais. In International Conference on Machine Learning, pages 35655–35739. PMLR, 2023.

[154] Thomas M Ward, Pietro Mascagni, Yutong Ban, Guy Rosman, Nicolas Padoy, Ozanan Meireles, and Daniel A Hashimoto. Computer vision in surgery. Surgery, 169(5):1253–1256, 2021.

[155] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. PhD Thesis, 1989.

[156] Michael W Whalen, Andrew Gacek, Darren Cofer, Anitha Murugesan, Mats PE Heimdahl, and Sanjai Rayadurgam. Your" what" is my" how": Iteration and hierarchy in system design. IEEE software, 30(2):54–60, 2012.

[157] Pierre Wolper. Constructing automata from temporal logic formulas: a tutorial. In School organized by the European Educational Forum, pages 261–277. Springer, 2000.

[158] Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. Nature, 602(7896):223–228, 2022.

[159] Botian Xu, Feng Gao, Chao Yu, Ruize Zhang, Yi Wu, and Yu Wang. Omnidrones: An efficient and flexible platform for reinforcement learning in drone control. IEEE Robotics and Automation Letters, 2024.

[160] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. Siren's song in the ai ocean: a survey on hallucination in large language models. arXiv preprint arXiv:2309.01219, 2023.

[161] Alwin Zweerink, Anne-Lotte CJ van der Lingen, M Louis Handoko, Albert C van Rossum, and Cornelis P Allaart. Chronotropic incompetence in chronic heart failure: a state-of-the-art review. Circulation: Heart Failure, 11(8):e004969, 2018.