Regular Transformations In Sequential Optimization & Reinforcement Learning

by

Taylor Dohmen

B.S., University of Mary Washington, 2017

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

Sriram Sankaranarayanan

Fabio Somenzi

Majid Zamani

Jyotirmoy Deshmukh

Dohmen, Taylor (Ph.D., Computer Science)

Regular Transformations In Sequential Optimization & Reinforcement Learning

Thesis directed by Professor Ashutosh Trivedi

Sequential optimization is a subfield of mathematical optimization—lying at the confluence of decision theory, operations research, game theory, artificial intelligence, and stochastic control—that focuses on problems involving notions of time and temporal dependence. Such problems center around an optimizing agent who exists within some larger environment and is assigned the task of obtaining a strategy for interacting with the environment in a manner that is optimal with respect to some underlying evaluation criteria. The canonical problem of interest in this area is that of finding a policy that maximize an expected payoff from a sequence of rewards generated by repeated interaction in Markov decision processes.

Regularity describes a wide-ranging notion of tractability from the study of formal languages and automata theory, which is exemplified by the family of regular languages. Due to a combination of convenient representability—as *e.g.* finite automata & regular expressions—and amenability to algorithmic methods, regular languages have long enjoyed adoption as modeling tools across a variety of disciplines within computer science including formal methods, natural language processing, software engineering, and more. Recently, regular languages have found significant roles in novel approaches to problems from sequential optimization related to non-Markovian dynamics and high-level logical objectives.

This dissertation focuses on extending the line of research at the intersection of formal languages and sequential optimization, starting with a generalization of regularity. We propose an abstract form of regularity that emphasizes a relational perspective and gives prominence to *transformations of regular languages*. After identifying a few types of transformations that are regular in this new sense, we explore their utility as modeling tools across a variety of aspects of sequential optimization. Our investigation yields a number of promising outcomes that include (1) advancements to existing approaches that

leverage regularity towards optimization in challenging types of environments such as those with non-Markovian dynamics and those with infinite state spaces, (2) original research directions stemming from the introduction of novel optimization objectives, and (3) insights around paths and obstacles to further applications of regular transformations in sequential optimization. This thesis presents both contributions to sequential optimization that are obtained through the use of regular transformations and contributions to the theory of regular transformations that should facilitate further applications of regularity in optimization.

Dedication

To Nadia & Kona.

Acknowledgements

To all those who supported me during this endeavor by generously giving their time and attention, I am deeply grateful.

- I thank my advisor, Ashutosh Trivedi, for his enthusiasm, patience, and guidance during my time at CU Boulder. His unwavering belief in my abilities kept me afloat in a sea of my own doubt.
- Thanks to all of my collaborators, who each helped mold me into the researcher I am today.
- Thanks to my lab mates and friends who seemed to always have time for long meandering discussions.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

Sequential optimization is subfield of mathematical optimization—lying at the confluence of decision theory, operations research, game theory, artificial intelligence, and stochastic control—that focuses on problems involving notions of time and temporal dependence. Such problems center around an optimizing agent who exists within some larger environment and is assigned the task of obtaining a strategy for interacting with the environment in a manner that is optimal with respect to some underlying evaluation criteria. The canonical problem of interest in this area is that of finding a policy that maximize an expected payoff from a sequence of rewards generated by repeated interaction with an environment. This thesis considers such problems in both the *planning* setting [Put94; FV96; FS01], where optimizing agents have full knowledge of their environment, and in the *reinforcement learning* setting [SB98], where the agent begins with no knowledge of its environment and must learn by observing outcomes of its actions.

Regularity roughly describes a wide-ranging notion of tractability from the study of formal languages and automata theory, which is exemplified by the family of regular languages. Due to a combination of convenient representability—as *e.g.* finite automata & regular expressions—and amenability to algorithmic methods, regular languages have long enjoyed adoption as modeling tools across a variety of disciplines within computer science including formal methods, natural language processing, software engineering, and more. Recently, regular languages have found significant roles in novel approaches to problems from sequential optimization related to non-Markovian dynamics and high-level logical objectives.

This dissertation proposes an abstract form regularity that emphasizes a relational perspective and gives prominence to *transformations of regular languages*. After identifying a few types of transformations that are regular in this new sense, we explore their utility as modeling tools across a variety of aspects of sequential optimization. Our investigation yields a number of promising outcomes that include

(1) applications of regular transformations towards optimization in challenging types of environments such as those with non-Markovian dynamics and those with infinite state spaces,

(2) results and original research directions stemming from new optimization objectives inspired by regular transformations, and

(3) insights around paths and obstacles to further applications of regular transformations in sequential optimization.

This thesis presents both contributions to sequential optimization that are obtained through the use of regular transformations and contributions to the theory of regular transformations that should facilitate further applications of regularity in optimization.

1.1 Thesis Statement

It is the position of this thesis that regularity encompasses more than regular languages. We argue in favor of a more general notion of regularity, with an emphasis on preserving desirable properties related to representation, computability, and stability. This dissertation nominates some families of *transformations of formal languages* that are regular in this sense and explores their utility in the area of sequential optimization.

Thesis Satement

Regular transformations form a versatile collection of computational tools for sequential optimization, facilitating

- effective representation & analysis of optimization processes and
- discovery of fundamental optimization problems and research questions.

1.2 Contributions & Organization

Following this introduction, Chapter 2 goes into a deeper discussion of a philosophical perspective around regularity, and introduces in detail the types of transformations that appear in subsequent chapters. Chapter 3 sets some notational conventions and provides technical definitions that are throughout the manuscript.

The main body of work presented in this thesis is divided into three complementary parts.

PART I focuses on relatively direct applications of language transformations and their models to solve problems in sequential optimization. This includes

(1) proposing *probabilistic reward machines* (PRMs) as a modeling tool for decision processes with non-Markovian reward dynamics, and

(2) formulating an approach using regular languages and rational transductions as a symbolic modeling tool for optimization and reinforcement learning in large and infinite-state *regular Markov decision processes* (RMDPs).

CHAPTER 4 introduces the model of probabilistic reward machines, establishes their theoretical foundation, and develops an inference algorithm for PRMs through a combination approach based on active automaton inference and model-free reinforcement learning.[1]

CHAPTER 5 adapts the modeling approach used by the verification framework known as *regular model checking* [Bou+00; Abd+04] to design a class of MDPs based on representing states with regular languages and state transitions as regular transformations. The main contributions here include (1) introducing of RMDPs and establishing foundational properties regarding decidability and approximability, and (2) *regular reinforcement learning*, an approach for reinforcement learning in RMDPs based the use of graph neural networks to approximate values of process states via their representation as finite automata.[2]

[1] Chapter 4 is based on an ICAPS 2022 paper [Doh+22].
[2] Chapter 5 is based on a CAV 2024 paper [Doh+24].

These contributions expand the scope of application of regularity within sequential optimization. Moreover, each contribution is obtained by adapting techniques that leverage properties or structure characteristic of regularity. This shows that our notion of regularity is both useful and faithfully, if approximately, preserves desired qualities of the regular languages. Both pieces of work contribute to sequential optimization, but in distinct aspects and with different methods. Furthermore, each approach takes advantage of significantly different aspects of regularity. This provides evidence in support of the claim that regularity supports versatility.

PART II presents a line of research that developed organically from the prospect of further extending the reward machine approach by using cost register automata as the basis for *register reward machines*. From considering models of quantitative regular transformations [Alu+13], arose the curious idea of discounting past outcomes in contrast to the typical notion of discounting possible future outcomes. *Past-discounting* is an exceptionally simple way to mathematically model a past-oriented time preference, and it has a natural interpretation from the sequential optimization perspective when past-discounted reward streams are thought of in terms of streams of incoming assets with depreciating values. The main contributions of this part center around several novel objective functions based on past-discounted reward sequences, which we introduce and study.

CHAPTER 6 contains initial theoretical developments for a model of register reward machines, through which the concept of past-discounting is introduced.

CHAPTER 7 poses an interpretation of past-discounting in terms of *depreciating assets* and studies discounted and limit-average optimization problems over depreciating asset streams. [3]

CHAPTER 8 defines novel objective functions based on plain limits of past-discounted reward sequences (rather than limits of discounted sums or averages of these reward sequences), and investigate solution methods for optimization under these new objectives.

This part supports our claim that, besides providing direct applications to optimization, regularity can provide new perspectives that lead to new discoveries and meaningful research questions.

[3] Chapter 7 provides an unabridged version of an AAMAS 2023 extended abstract [DT23].

PART III contains work that makes progress on bridging the remaining gaps that separate more powerful notions of regular transformations from further uses in sequential optimization. These are contributions that extend and solidify portions of the theory around regular transductions and transducers. First, we introduce and study nondeterministic variations of monadic second-order transducers [CE12] and streaming string transducers for infinite strings [AD11] (ω-NMSOTs and ω-NSSTs). Next, we look towards generalizing the regular Markov decision processes from Part I by adapting the approach of regular model checking to enable verification of infinite-state systems with transition relations representable by these models. Lastly, we fill some gaps in the literature around composition of streaming string transducers that were found during the process of extending regular model checking.

CHAPTER 9 defines ω-NSSTs and ω-NMSOTs as models of ω-regular transductions and proves their equivalence.[4]

CHAPTER 10 presents an adaptation of regular model checking to allow for the use of (ω-)regular transductions. An important part of this contribution is establishing decidability of the type checking problem for ω-NSSTs.[4]

CHAPTER 11 repairs a construction for SST composition, originally due to [AD11], which was found to have a bug. We provide a revised presentation of the original construction, followed by additional analyses explaining the nature of the bug as well as proposing an augmentation to the procedure that recovers the construction by adding a post-processing step.

Contributions In Context

In this section, we review the most relevant related literature and position the contributions of this dissertation within the context of this body of work. The following paragraphs briefly survey some major threads of inquiry at the intersection of regularity and sequential optimization.

Reward Machines. Reward machines [Ica+18; Ica+22; Ica22] have arisen as a promising approach to modeling history-dependent reward dynamics in certain kinds of non-Markov decision

[4] Chapters 9 and 10 are based on an FCT 2021 paper [Dav+21b].

processes. A reward machine is essentially a finite-state machine that reads state-action sequences generated by an agent interacting with a decision process and outputs the reward values used in the optimization process. Highlights of the research into reward machines include algorithms for learning reward machines in partially observable decision processes [Ica+19], methods for jointly learning reward machines and corresponding optimal policies [Xu+20], and adaptations of active inference algorithms for learning unknown reward machines [Xu+21].

Regular Decision Processes. Regular decision processes [BG19c; BG19b] extend the approach of reward machines to modeling all of the dynamics of a decision process. A regular decision process capture weakly non-Markov decision processes, in which state-transition probabilities and reward dynamics are regularly history-dependent. As opposed to general non-Markov decision processes, which allow for arbitrary history dependence, the non-Markovian dynamics of regular decision processes are restricted such that they can be simulated via finite-state machines.

Omega-Regular Objectives. The final body of related work we will discuss for now concerns the introduction of novel objectives in sequential optimization. Much of the research around this topic is related to ω-regular objectives [Hah+19a; Hah+19b; ST19]. These are optimization objectives formulated in terms of ω-regular languages. In this context, an ω-regular language is interpreted as something like a target for the optimizing agent. The intention is that the infinite sequence of interaction that the agent generates will be contained within this ω-regular target in the limit. To facilitate this asymptotic outcome, the primary technique [Hah+20; Kaz+22] is to construct a reward signal that guides the agent towards meeting the ω-regular specification with maximal probability.

Figure 1.1 displays a visual depiction from a bird's eye view of the state of the relationship between regularity and sequential optimization, including all of the related work mentioned in this section.

Figure 1.2 shows a new version of the map from Figure 1.1, in which the contributions of this thesis are added. The intent of the figure is twofold: (1) to clarify how the work presented here relates to work from the literature and fits into a greater context. (2) to delineate the portions of our work that bridge the gap between automata theory and sequential optimization (the bold labels positioned across

the boundary in the blue region) vs. the portions of our work that make progress towards enabling the adoption of regular transformations in sequential optimization, but need further development to have meaningful impact (the bold label positioned in the red region.) The dotted arrows connecting the labels on this map indicate influence. For example the arrow going from “rational transductions” to “regular transductions” represents that regular transductions arose from the earlier concept of rational transductions.

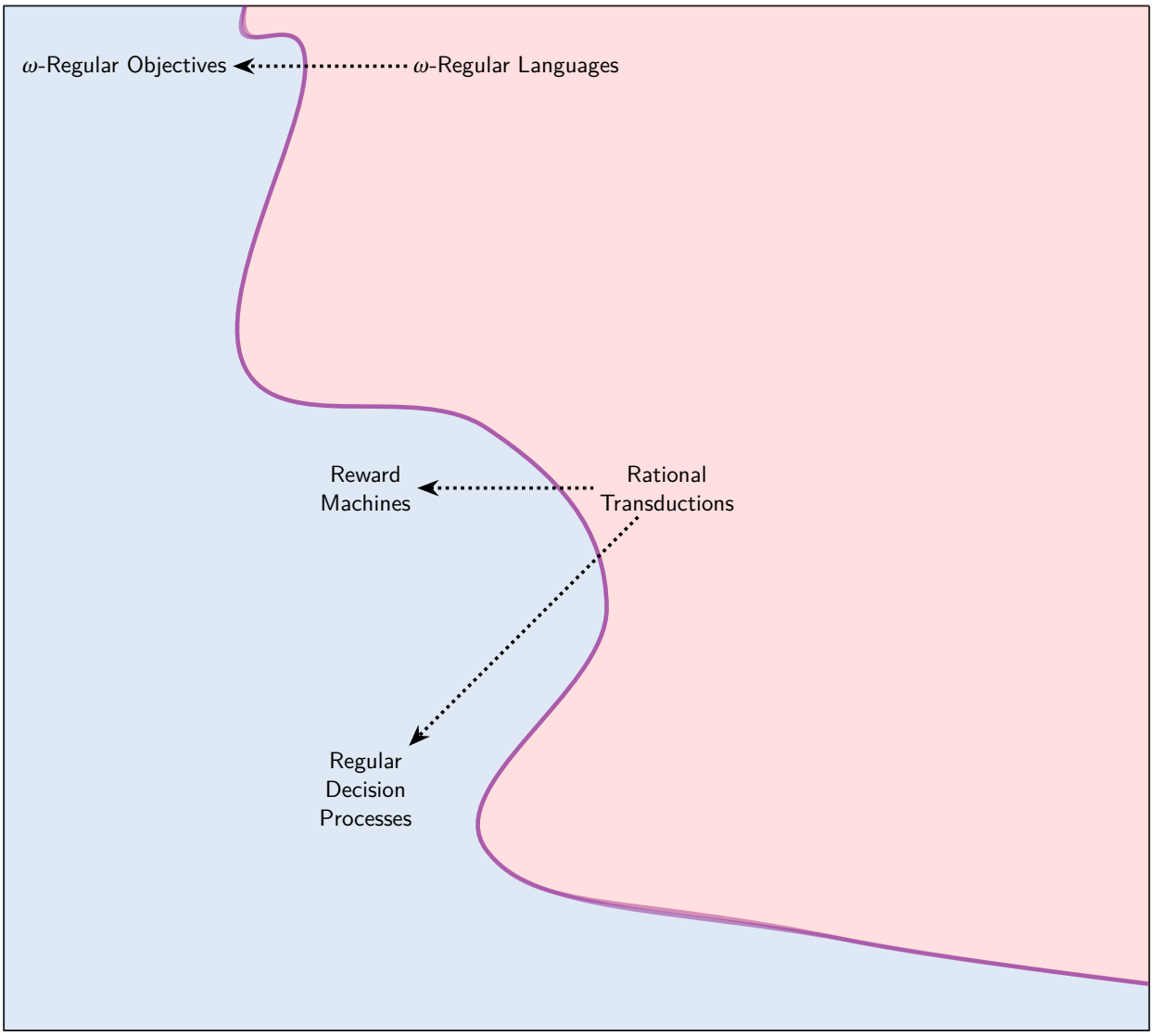


Figure 1.1: A map of regularity's reach into sequential optimization **before** this dissertation.

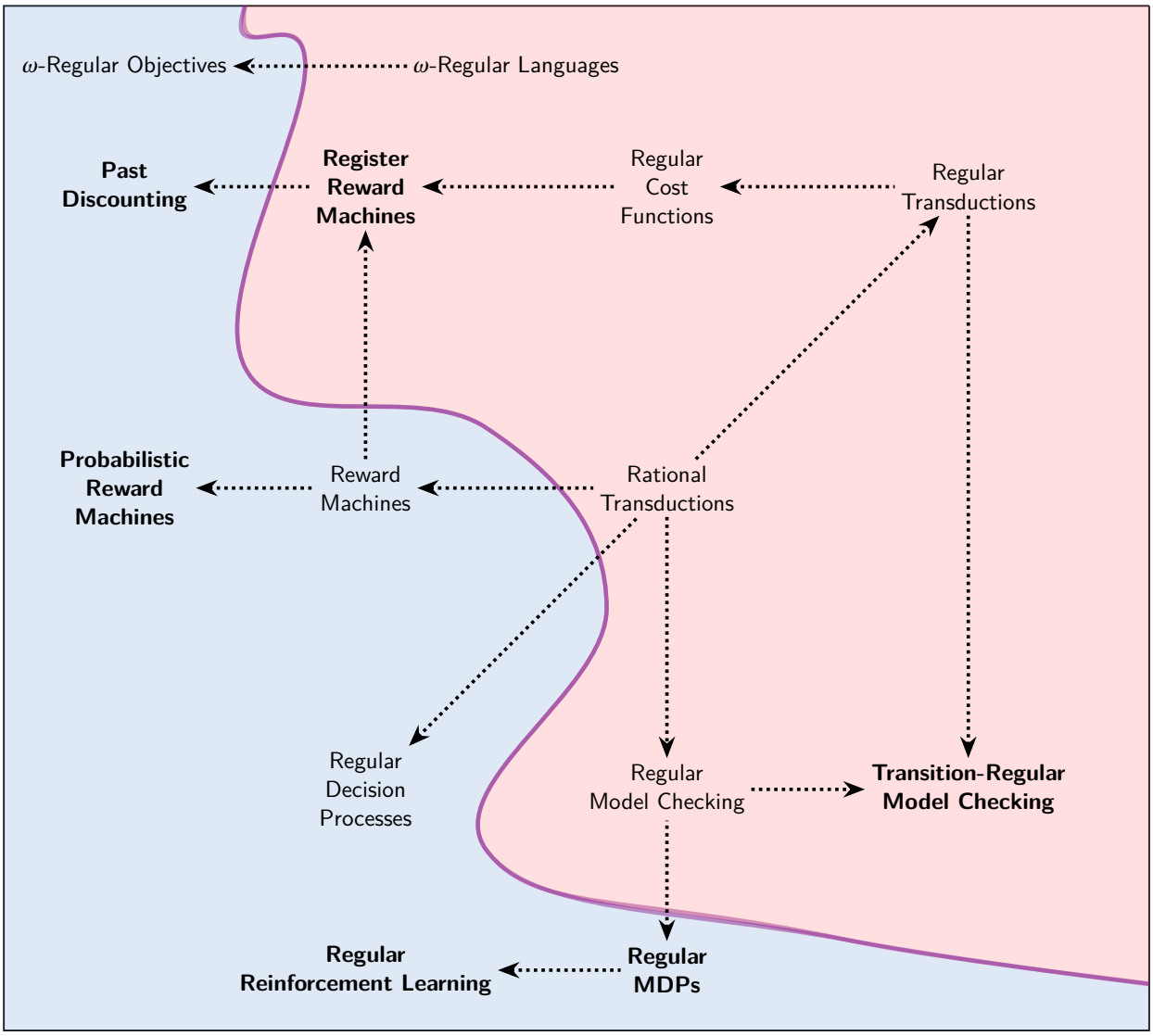


Figure 1.2: A map of regularity's reach into sequential optimization **after** this dissertation.

Chapter 2

Regularity — From Languages To Transformations

Regular languages—lying at the bottom of the classic hierarchy of formal languages

$$\text{Regular} \subset \text{Context-Free} \subset \text{Context-Sensitive} \subset \text{Recursively Enumerable}$$

laid out by Chomsky [Cho59]—comprise, more or less, the simplest class of interesting languages. Most commonly, they are defined as the languages that can be recognized by finite-state automata or that can be described by regular expressions. The word *regular*[1] hints at some characteristics that make these languages useful.

(1) In the sense of *usual* or *typical*: the regular languages are precisely captured by a surprising number of distinct formal definitions, and they enjoy strong closure properties, *cf.* Table 2.1[2] .

(2) In the sense of *orderly* or *systematic*: regular languages are sufficiently structured for many of their non-trivial properties to be algorithmically decidable, *cf.* Table 2.2.

From our perspective, these are two core aspects of regularity that contribute to the success of applications in areas like formal verification.

The *ω-regular languages* [PP04] form a family of ω-languages—sets of infinite-length strings—analogous to the regular languages. Despite being slightly more subtle in nature, the ω-regular languages

[1] The name *regular* is attributed to Kleene [Kle56], who intended it to be temporary until a more suitably descriptive name was proposed. An alternative term did not come to be, and so *regular* became the de facto title for this language class, adding one more drop to the ocean of things dubbed *regular*, *normal*, *etc.* by mathematicians and scientists.

[2] In Table 2.1, green cells indicate that the class of languages is closed under the operation, while red cells indicate the opposite. In Table 2.2, green cells indicate that the problem is decidable for the language class, while red cells indicate undecidability. Results are compiled from common references [HMU07; Sip97; Sak09].

	Union	Intersection	Complementation	Morphism
Recursively Enumerable	✓	✓	✗	✓
Context-Sensitive	✓	✓	✓	✗
Context-Free	✓	✗	✗	✓
Regular	✓	✓	✓	✓

Table 2.1: Closure properties of various language classes.

	Membership	Emptiness	Universality	Equivalence	Inclusion
Recursively Enumerable	✗	✗	✗	✗	✗
Context-Sensitive	✓	✗	✗	✗	✗
Context-Free	✓	✓	✗	✗	✗
Regular	✓	✓	✓	✓	✓

Table 2.2: Decidability properties of various language classes.

also lie at the confluence of many different definitions and enjoy, more or less, the same desirable closure properties and decidability properties indicated for regular languages in Tables 2.1 and 2.2. Unless explicitly excluded, we shall proceed in including the ω-regular languages implicitly when discussing the regular languages.

2.1 A Philosophical Perspective On Regularity

The fruitful research around applications of regular languages in sequential optimization demonstrates that regularity is indeed a meaningful and useful concept in sequential optimization. It is our position, however, that the regular languages are not the end-all-be-all of regularity, and we posit that they are actually representative of a more general notion of regularity. In this section, we provide a more detailed review of the theory around regular languages, focussing on highlighting a few key principles that are then used as the foundation of an abstract notion of regularity that applies not only to languages but to their transformations. We believe that broadening the scope of regularity itself can broaden the

scope of its utility in sequential optimization.

Generalizations of regular languages tend to fall on two distinct axes. In one direction, we find families of formal languages that subsume the regular languages. As noted earlier, however, many of the desirable properties of the regular languages are already lost when moving up to the next level of Chomsky's hierarchy to the context-free languages, making the outlook with respect to preserving regularity in this direction fairly bleak. That is not to say that this path leads only to disappointment[3] , but that a number of unavoidable barriers to generalization exist along this dimension.

In the other direction, we encounter classes of *transformations with regular domains*. These are functions and relations that map regular languages to arbitrary codomains, anchoring them to regularity in a quite literal sense. It is in this dimension that we shall explore in the context of sequential optimization. In following this path, we adopt a relative perspective, emphasizing relationships as opposed to entities.

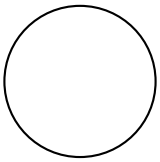
2.1.1 A Relative Viewpoint

The title of this subsection is worded intentionally to hint at a source of inspiration: *Grothendieck's relative viewpoint*, a heuristic which posits, roughly, that mathematical objects should be understood relative to their place within some greater context. This attitude has proved revolutionary in a variety of fields of mathematics[4] , and, while we do not pretend to achieve anything so grand in the present work, we attempt to adopt this basic philosophy in the hope of moving towards some higher insight or understanding.

Funnily enough, the relative viewpoint also provides a useful conceptual framework for the process of abstraction that we are about to undertake. To understand this framework, let us forget about regularity for a moment, and consider the more familiar concept of circularity. Most people have some basic intuition about what it means to be circular, involving resemblance to the following shape.

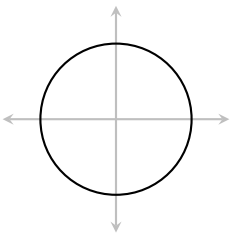
[3] More general language classes resembling the regular languages have been found, including visibly pushdown languages [AM04; Alu+05] and operator-precedence languages [CMM78; Pan16; Pan+13; Hen+23].

[4] Perhaps most notably, and as a direct result of Grothendieck's specialization in these areas, the relative viewpoint, or the functorial perspective as it is sometimes called, forms a central tenet of category theory and modern algebraic geometry.

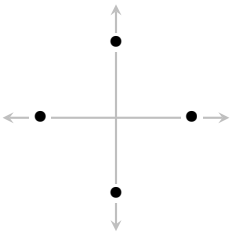


But a precise definition of the notion of circularity is elusive, nonetheless. In what capacity and to what extent must something be similar to this idealized circle to be considered circular?

One approach is to begin with a concrete object that we imagine to be a faithful representation of the abstract form of a circle. For instance, one might choose such a representation to be the euclidean unit circle in the cartesian plane, depicted as follows.



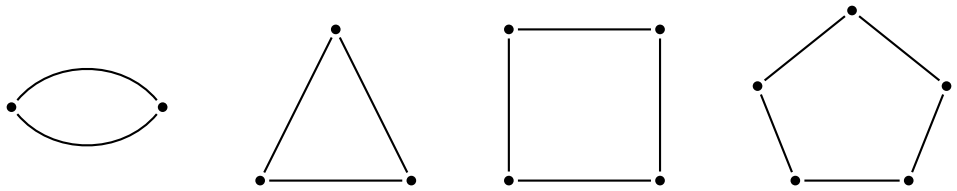
This unit circle consists of the set of all points in the plane that are 1 unit away from the origin in any direction, *i.e.* the set $\{\langle x, y\rangle \in \mathbb{R}^2 : x^2 + y^2 = 1\}$. Common intuition might suggest that four discrete points in the euclidean plane



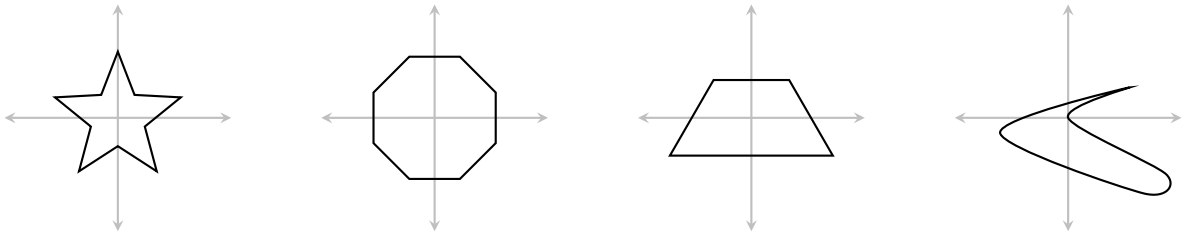
do not constitute a circle, but these points comprise the totality of *integer* points in the plane that are 1 unit away from the origin, *i.e.* the set $\{\langle x, y\rangle \in \mathbb{Z}^2 : x^2 + y^2 = 1\}$. Despite diverging quite sharply from our intuition, the similarity of the integer unit circle to the real unit circle—that they are each the set of solutions to the equation $x^2 + y^2 = 1$ for their respective number types—raises a strong argument in favor of describing this collection of four points as circular.

One possible interpretation of this dissonance is that we simply started with an inappropriate initial representative of circularity in the unit circle. Perhaps we leaned too hard on algebraic tools and notions of distance, and we should begin with a less rigid formulation. To test this hypothesis, one might

adopt a topological perspective, which need not consider notions such as distance. One option is to assert that circularity should require some notion of connectedness. Requiring connectedness between points of a circle excludes the set of four isolated points above, but not connected discrete graphs, such as any the following.



Alternatively, we could again begin with the real unit circle and assert any continuous deformation of this unit circle to be circular. This definition includes virtually every type of form used as an example of a non-circle in classic euclidean geometry, such as any of the following.



What this suggests is that our initial attempt to define circularity did not excessively rely on algebraic or geometric ideas, but that any general notion like circularity will inevitably include some things that do not seem to fit with our preconceived ideas.

All of this is to drive home the point that, like circularity, *regularity is a polymorphic notion.* It is an abstraction used to classify things as being qualitatively similar to the regular languages in some aspect, but whatever this aspect might be is subject to change. Though the topic of discourse will be quite different—formal languages and their transformations as opposed to geometric shapes—we shall treat regularity analogously to how we have treated circularity in this section. Each manifestation of regularity emphasizes and sacrifices some particular qualities of the regular languages, and each variation can be useful in an appropriate situation. These sacrifices are the price of doing business with generalization, but the cost is not sunk; it is an investment in an abstraction, and the return will be measured by the fruits of its application.

2.1.2 The Crux

It is now time to expound upon the nature of the regular languages, with a view towards extracting some essential aspects of regularity.

As remarked upon at the beginning of the chapter, the regular languages are precisely captured by a number of different formal definitions of varying style, including characterization as the class of languages that:

- are recognized by deterministic [Huf54; Moo56; CM58], nondeterministic [RS59], two-way [RS59; She59], and alternating [CS76; CKS81] finite-state automata[5] ;
- are definable in monadic second-order (MSO) logic[6] [Büc60; Elg61; Tra62a; Tra62b];
- have finite syntactic monoids[7] [Myh57; Ner58];
- are specified by regular expressions [Kle56; CEW58; MY60];
- are generated by right-linear and left-linear grammars[8] [CS59];
- coincides with the complexity class of languages decidable in constant space: $\mathsf{DSPACE}(O(1))$.

This list is not exhaustive, but it is large enough to convey that there are a multitude of ways to capture these languages, and it is broad enough to provide a reasonably representative sample of what sorts of formalisms are typical in this setting.

The characterizations of regular languages are often categorized into three groups based on their bases in automata theory, logic, and algebra [FR16]. This division is informative, but ambiguous and incomplete. To see the ambiguity, consider the algebraic category of models. The algebraic approach is often championed by the Myhill-Nerode theorem, based on syntactic monoids which are abstract algebraic structures. Regular expressions are also undeniably algebraic in nature, but are often left out of this category. Granted, regular expressions have quite a different flavor compared to syntactic monoids, but they certainly belong somewhere in this classification that currently leaves them out. The

[5] ω-regular languages are recognized by ω-automata [Mul63; McN66; Tho90; Tho97].
[6] Infinite strings are considered already in the original work of Büchi [Büc60].
[7] The characterization of regular languages in terms of finite syntactic monoids is commonly known as the Myhill-Nerode theorem. This approach has been extended to ω-languages [Arn85].
[8] Grammars that are either right-linear or left-linear, but not both, are called regular grammars.

incompleteness of the taxonomy is witnessed by the regular grammars. It is difficult to, in good faith, consider grammars as being algebraic, logical, or automata-theoretic, and so they are usually omitted from this classification in which they deserve a place.

In this work, we offer an adaptation of this triadic categorization that is a bit less specific and a bit more inclusive. We classify characterizations of the regular languages using an alternative set of descriptors: *imperative*, *declarative*, and *abstract*. The terms "imperative" and "declarative" are borrowed from the discourse around programming paradigms [ABZ92]; here, they are qualifiers related to the representation of regular languages. The term "abstract" is intended to describe those characterizations that are divorced from any particular representation.

The Imperative

Imperative programming is an *operational* approach to algorithm design that is oriented around action. An imperative program is a sequence of instructions that a computer should follow such as assigning a value to a variable, opening a file, or changing the color of a pixel. Formalisms that capture the regular languages imperatively include the various kinds of automata and grammars listed above. Automata represent regular languages by providing a means of recognizing their individual elements. The word "recognize" emphasizes the perspective that an automaton is an active entity that performs an analysis on its inputs to classify them as belonging to a language or not. Grammars are also imperative representations, but they are synthetic rather than analytic. A grammar stipulates a set of actions that may be combined in sequence to generate strings of a language, and the word "generate" highlights their action-oriented nature.

The Declarative

Declarative programming is a *descriptive* approach to algorithm design that is oriented around outcome. A declarative program is a specification that describes the intended results of a computation, but does not explicitly lay out the steps that should be taken by the computer to achieve such a result. Regular languages are represented declaratively by algebraic expressions and logical formulas. Both reg-

ular expressions and formulas in MSO logic specify the syntactic form of strings belonging to languages, but neither expressions nor formulas directly facilitate procedures for deciding whether a given string belongs to or generating strings from the languages they describe.

Remark 2.1

The imperative and declarative paradigms complement each other by providing opposite perspectives regarding form and function. Imperative representations emphasize function and elucidate process. Declarative representations emphasize form and illuminate structure.

The Abstract

Abstract characterizations of the regular languages are *model-independent*. While both imperative and declarative approaches to representation are abstract, their concrete manifestations—*e.g.* MSO formulas, regular expressions, automata, *etc.*—are fundamentally syntactic. The abstract perspective complements them both with a more semantically oriented view. Abstract approaches for capturing the regular languages include the Myhill-Nerode theorem, which captures regular languages in terms of finitary congruence classes of syntactic monoids, and the identification of regular languages with the complexity class of problems that are decidable in constant space. By defining regular languages in a manner that is decoupled from any particular system of representation, semantic perspectives allow for more direct inquiry into the position of regularity in the context of formal languages.

Imperative	Declarative	Abstract
Finite-State Automata	Monadic Second-Order Logic	Finite Syntactic Monoids
Regular Grammars	Regular Expressions	DSPACE($O(1)$) Computability

Table 2.3: Classification of various characterizations of the regular languages.

This proposed categorization—summarized in Table 2.3—of the most fundamental characterizations of the regular languages demands justification. Accordingly, let us examine the delineations that this particular taxonomy illuminates and explain their significance with regard to applications in

sequential optimization.

Imperative models facilitate algorithmic efficiency. As mentioned earlier, an important feature of the regular languages is their decidability properties. Amongst the plethora of formalisms capturing regular languages, imperative models—and automata in particular—stand out as those best suited for algorithmic manipulation. By providing graphical representations, automata enable the treatment of regular languages from the perspective of graph theory. Since graphs and graph algorithms hold a prominent position in the field of algorithm design and analysis, automata allow this extensive body of work to be applied to regular languages. As an example, emptiness of a regular language can be decided in logarithmic space by applying reachability algorithms to the graph of its representation as a deterministic finite-state automaton. Speaking of determinism, automata also make explicit the dichotomy between determinism and nondeterminism, which is somewhat hidden by other kinds of models. Even though nondeterminism does not lead to additional expressive power, it does tend to result in increased complexities for decision problems. This distinction becomes even more significant when considering more powerful kinds of abstract machines, where decidability properties are often contingent upon the determinacy of models.

Declarative models support compositionality and conciseness. Declarative models collectively comprise the meta language used to describe regular languages. Formalisms such as MSO logic and regular expressions provide a means of expressing regular languages symbolically, much in the same way that written words and sentences symbolically express natural language. Aspects such as determinism in relation to processes of recognition or generation are hidden away by these models, making these interfaces more suitable for human reading and writing. Formulas and expression may be easily combined by operations such as chaining and nesting, thereby facilitating compositionality in a straightforward manner. Moreover, these representations are dense in information, making them concise and compact. This compactness comes at the cost of algorithmic tractability, exemplified by the fact that converting an MSO formula into an equivalent automaton produces a machine with a

non-elementary number of states in the size of the formula[9] .

Abstract characterizations expose limitations and provide explanations. Whereas imperative and declarative models are useful for capturing regularity, abstract characterizations of the regular languages are useful in exposing irregularity. In order to understand and take advantage of the regularity present in the regular languages, it is essential for us to understand their limits and to contextualize them relative to that which lies outside their reach. Both examples of abstract characterizations discussed so far—the Myhill-Nerode theorem and the identification with $\mathsf{DSPACE}(O(1))$—provide semantic criteria illuminating the boundary of the regular languages. Each facilitates an explicit method for proving a language to be irregular: the former via establishing a language has an infinite syntactic monoid, and the latter by showing a $\log\log n$ lower bound[10] on the language's space complexity. Additionally, abstract characterizations can be a source of inspiration and explanation for developing and analyzing algorithms that work directly with imperative or declarative models. Two pieces of evidence for this claim are found in (1) Hopcroft's [Hop71] automaton minimization algorithm and (2) Angluin's [Ang87] $\mathrm{L}^\star$ algorithm for automaton inference, both of which heavily lean on the theoretical foundation ensured by the Myhill-Nerode theorem.

2.2 Regularity For Transformations

We now nominate some classes of transformations as representatives of the general notion of regularity outlined in this chapter. The intent of this section is to portray a landscape of regularity for transformations by introducing and contextualizing the concrete types of transformations and models that appear in this thesis. In particular, we shall fix our attention towards two broad categories of transformations over the regular languages:

- *transductions*, which are mappings between formal languages and
- *quantitative transformations* that map from formal languages strings to sets of numbers.

[9] For formulas of size n, there is no k-tuply exponential function, for any fixed k,—*i.e.* of order $O\left(\underbrace{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}_{k \text{ times}}\right)$—that bounds the size of the resulting automaton [MS73; Mey06; Sto74; SM02].

[10] Stearns, Hartmanis, and Lewis [SHL65] show that non-regular languages have at least doubly logarithmic space complexity by establishing that $\mathsf{DSPACE}(O(1)) = \mathsf{DSPACE}(o(\log\log n))$.

2.2.1 Transductions

A *transduction* , generally speaking, is a binary relation $\theta : \Sigma^* \times \Gamma^*$ between strings over some finite alphabets Σ and Γ, and a *transducer* is a model of a transduction in much the same way as a matrix is a model of a linear transformation or as a Turing machine is a model of a computation. The focus of the present work are some of the subclasses of transductions that can reasonably be considered regular. While "regular languages" has become standard terminology in computer science, no class of transductions has undisputed claim to the name "regular". For now, it suffices to introduce some of the types of transductions that are good candidates for the title. The only explicit requirement we impose is that the domain of a regular transduction should be a regular language.

Rational Transductions

The simplest class of transductions we consider are the *rational transductions* [Eil74a; Ber79], which are best introduced by describing their most common models. A *finite-state transducer*[11] (FST) is a type of abstract state-machine that is obtained by augmenting the classic finite-state automaton model by adding an output alphabet and allowing the machine to append a finite string to an output tape upon each transition. That is, every transition of an FST has the form $q \xrightarrow{\sigma\backslash w} q'$, which is read as "from state q, if the symbol σ is read from the input tape, then the string w is appended to the output tape, and the state changes to q'". If the process ends in an accepting state, then the pair $\langle x, y \rangle$, where x is the string on the input tape and y the string on the output tape, is an element of the transduction modeled by the machine. Figure 2.1 depicts a finite automaton next to an FST obtained by augmenting it with this kind of output capability. The automaton recognizes the language of binary strings with an odd number of 1 symbols. The transducer maps each such string to a string over the alphabet $\{a, b\}$ by substituting every 0 symbol by the symbol a and every 1 symbol by the string bb.

[11] Finite-state transducers are often referred to by other names in the literature, especially in older papers. The most common alternative title is *generalized sequential machines* [GR66]. Other titles that may be used in reference to identical or slightly different models include *Moore machines* and *Mealy machines*, in acknowledgement to the early work of Moore [Moo56] and Mealy [Mea55], respectively.

(a) A finite-state automaton.

(b) A finite-state transducer.

Figure 2.1: A finite-state automaton and related finite state-transducer.

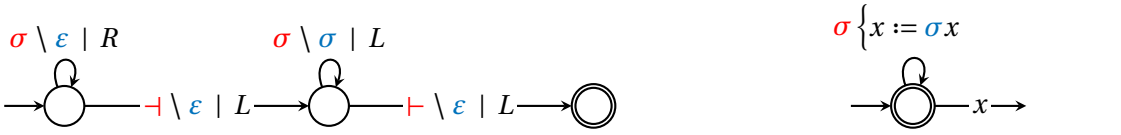
Regular Transductions

Despite the earlier discussion regarding the nonstandard use of the title "regular" when it comes to transductions, we adopt the terminology[12] for the next type of transduction we discuss: the *regular transductions* .

Perhaps the most natural way to arrive at the regular transductions is via a generalization of *deterministic finite-state transducers* (DFTs), which model[13] the *regular functions, i.e.* the functional subset of regular transductions. A classic early result from automata theory establishes that finite-state automata do not gain any expressive power from having the ability to move their reading heads back and forth over their input strings [RS59; She59]. That is, two-way finite-state automata capture exactly the regular languages, no different from their one-way counterparts. This is not the case, however, for transducers, with *Two-way finite-state transducers* (2FSTs) being considerably more powerful than one-way FSTs. It is easy to see this as a fact by considering the function $w \mapsto \overleftarrow{w}$ that maps strings to their reversals, *i.e.* $\sigma_1\sigma_2...\sigma_n \mapsto \sigma_n\sigma_{n-1}...\sigma_1$. There is no FST that can reverse its input string, but the relatively simple 2FST depicted in Figure 2.2 can. Each transition of the 2DFT has the form $q \xrightarrow{\sigma \setminus w | D} q'$, which is read as "from state q, if the symbol σ is read from the input tape, then the string w is appended to the output tape, the input head moves in direction $D \in \{L, R\}$ (either left of right), and the state changes to q'." The symbols $\vdash$ and $\dashv$ are markers for the ends of the string on the input tape, and ε represents the empty string. The class of functions modeled by *deterministic* two-way finite-state transducers (2DFTs)

[12] While not standard, "regular" has become a more common title for this class in recent years as additional models have been introduced capturing precisely these transductions. We adopt this language primarily as a convenient way to refer to these transductions without referencing any particular model.

[13] DFTs are models of rational functions, but they do not completely characterize this class of transformations. There are rational functions that cannot be implemented by a DFT, but that can be represented by a *functional* NFT.



(a) A two-way finite-state transducer.

(b) A streaming string transducer.

Figure 2.2: A 2FST and an SST, each implementing the reverse function $w \mapsto \overleftarrow{w}$.

are the regular functions.

Another model of the regular functions arose from a separate line of research on graph transformations definable in monadic second-order logic [Cou92; Cou94]. It was shown by [EH99; EH01] that *deterministic monadic second-order transducers* (DMSOTs) when restricted to string graphs—graphs where each vertex corresponds to a position of a symbol in a string and is limited to having at most one outgoing edge—define precisely the same functions captured by 2DFTs. Somewhat surprisingly, they also proved that the string relations definable by *nondeterministic monadic second-order transducers* (NMSOTs) and the relations modeled by *nondeterministic two-way finite-state transducers* (2NFTs) do not coincide and are incomparable[14] .

Even more recently, a third line of work [AČ10; AD11] introduced an alternative generalization of FSTs called *streaming string transducers* (SSTs). Just like FSTs, SSTs make a single pass over their inputs, but they have a different output mechanism. Instead of appending a word to an output tape on each transition, an SST is equipped with a finite set of write-only string variables that it updates on each transition. Alur and Černý [AČ10] showed that deterministic streaming string transducers (DSSTs) model the same functions as 2DFSTs and as DMSOTs, and Alur and Deshmukh [AD11] established that nondeterministic streaming string transducers (NSSTs) capture the same class of relations as the NMSOTs. Yielding to the majority, we call the class of transductions agreed upon by SSTs and MSOTs the *regular transductions*[15] .

[14] There are relations definable by an NMSOT that cannot be implemented by a 2NFST, and there are relations modeled by 2NFTs that cannot be defined by any NMSOT.

[15] Since the distinction is relevant in a portion of this thesis, we mention that SSTs capture the regular transductions only if the *copyless restriction* is implicitly assumed. The copyless restriction prohibits a variable to be included in the updated value of more than one variable in a single transition. A detailed discussion of variable copying is provided in Chapter 11. In a few places, we use the term *quasi-regular* to describe the transductions definable by *copyful* SSTs.

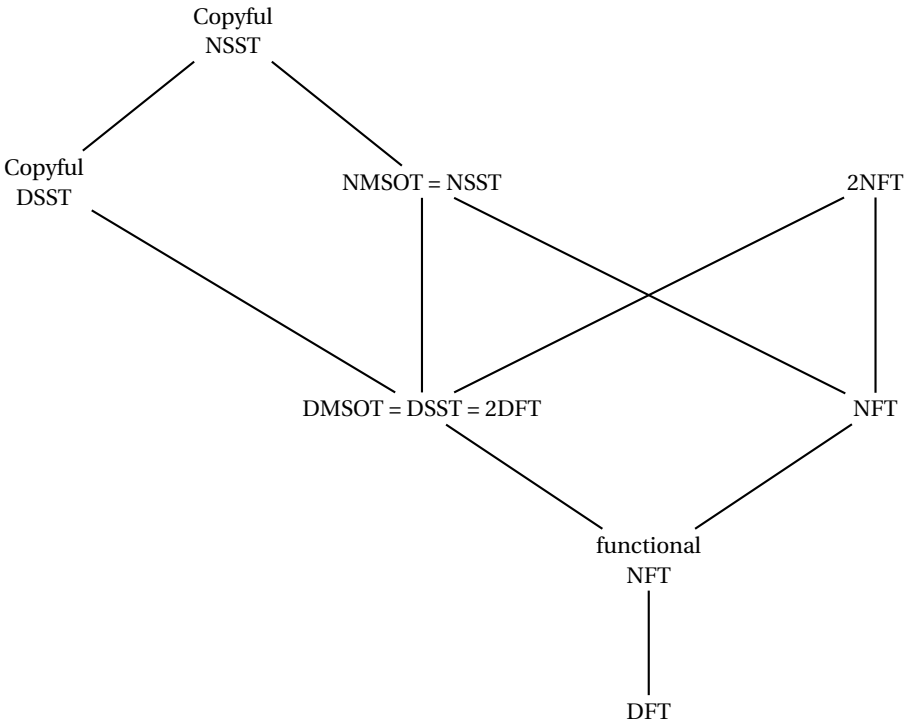


Figure 2.3: A Hasse diagram of the landscape of expressiveness with respect to various classes of transducers.

Figure 2.3 illustrates the hierarchy of expressiveness amongst the classes of transductions we have described in this section.

Tables 2.4 to 2.6[16] compile the status of relevant classes of transductions regarding the three kinds of characterizations outlined in this section, as well as the closure and decidability properties of these classes.

[16] For decision problems, green cells indicate that the given problem is decidable for the corresponding class of transductions, while red cells indicate the opposite. For closure properties, green cells indicate that the corresponding class of transductions is closed under the given operation, while red cells indicate the opposite.

Characterizations of Transduction Classes

	Imperative	Declarative	Abstract
Rational Functions	fNFT[a], Bimachines[b]	Order-Preserving DMSOT[c]	Finite-Index Congruences[d]
ω-Rational Functions	ω-fNFT	Order-Preserving ω-DMSOT	?
Rational Relations	NFT[e]	Order-Preserving NMSOT	?
ω-Rational Relations	ω-NFT, ω-Bimachines[f]	Order-Preserving ω-NMSOT	?
Regular Functions	DSST[g], 2DFT[h]	DMSOT[h i], RCE[j], RTE[k]	Finiteness Preserving Transducer Semigroups[l]
ω-Regular Functions	ω-DSST^m[m], ω-2DFT^{mk}	ω-DMSOT^m, ω-RTE^k	?
Regular Relations	NSST^n[n]	NMSOT^{hi}	?
ω-Regular Relations	ω-NSST^o[o]	ω-NMSOT^o	?
Quasi-Regular Functions	copyful DSST^p[p]	?	?
Quasi-Regular Relations	copyful NSST^p	?	?

[a] Elgot and Mezei [EM65]
[b] Schtzenberger [Sch61] and Eilenberg [Eil74b]
[c] Filiot [Fil15], Filiot, Gauwin, and Lhote [FGL19], and Filiot and Reynier [FR16]
[d] Reutenauer and Schutzenberger [RS91]
[e] Rabin and Scott [RS59]
[f] Filiot, Gauwin, Lhote, and Muscholl [Fil+18]
[g] Alur and Černý [AČ10]
[h] Engelfriet and Hoogeboom [EH99; EH01]
[i] Courcelle [Cou92; Cou94]
[j] Alur, Freilich, and Raghothaman [AFR14]
[k] Dave, Gastin, and Krishna [DGK18; DGK22]
[l] Bojanczyk and Nguyên [BN23]
[m] Alur, Filiot, and Trivedi [AFT12]
[n] Alur and Deshmukh [AD11]
[o] Dave, Dohmen, Krishna, and Trivedi [Dav+21b]
[p] Filiot and Reynier [FR17; FR21]

Table 2.4: Characterizations of various classes of string transductions.

Closure Properties of Transduction Classes

	Composition
Rational Functions	[EM65]
Rational Relations	[EM65]
Regular Functions	[Cou92; AČ10; CJ77]
Regular Relations	[Cou94; AD11]
2NFT-Definable Relations	[EH99; EH01]
Quasi-Regular Functions	?
Quasi-Regular Relations	?

Table 2.5: Closure properties of various classes of transductions.

Decidability Properties of Transduction Classes

	Functionality	Equivalence	Type Checking
Rational Functions	[BH77; DKR82; Cul79]		
Rational Relations	[BH77; DKR82]	[Gri68]	
Regular Functions		[Gur80; Gur82; AČ11]	[AČ11]
Regular Relations	[AD11]		[Dav+21b]
2NFT-Definable Relations	[EH07; EH01]		?
Quasi-Regular Functions		[FR17; FR21]	?
Quasi-Regular Relations	[FR17; FR21]		?

Table 2.6: Decidability properties of various classes of transductions.

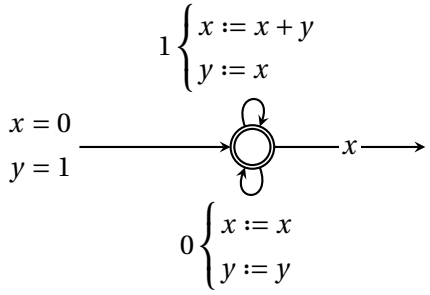


Figure 2.4: A CRA over the domain $\langle \mathbb{N}, + \rangle$ that defines the function $w \mapsto f(|w|_1)$ where $|w|_1$ is the number of 1 bits in w and $f(n)$ is the n^{th} Fibonacci number.

2.2.2 Quantitative Transformations

Quantitative transformations of regular languages are mappings $\Sigma^* \to \mathbb{D}$ from strings over a finite alphabet Σ to elements of a numerical domain $\mathbb{D}$ such as the integers or the real numbers. Since the quantitative transformations studied in this thesis are based on generalizations of the types of transducers described in the previous subsection, a brief account of their distinguishing features will suffice.

Probabilistic Automata

The first sort of quantitative transformations of concern in this manuscript are those that map words to probability distributions. The models capturing regular classes of such transformations are known as *probabilistic automata* and *probabilistic transducers*. These models are obtained by generalizing the model of finite-state transducers to include stochastic dynamics, and the resulting machines are useful in representing processes that are subject to noise or uncertainty.

Regular Cost Functions

The second sort of quantitative transformations we consider are known as *regular cost functions* , and are captured by *cost register automata* (CRA) [Alu+13], a model closely related to streaming string transducers[17] . CRAs have the same basic structure as SSTs, in that they read input strings in one direction

[17] Just as regular transductions are modeled by *copyless* SSTs, regular cost functions are modeled by *copyless* CRA. Copyful CRA model a strictly larger class of functions.

while maintaining and modifying a finite set of write-only variables. The difference is found in the data types that CRA hold and manipulate with their registers. Whereas SSTs only ever store strings, CRA are parameterized by a *cost model*. A cost model sets a cost domain such as the integers or the rational numbers, and provides a specification of allowed operations on elements of this domain stored in the registers of a CRA. When a CRA processes a string, it manipulates elements of its cost domain using the permitted operations, constructing terms over a term algebra instead of strings over an alphabet.

Chapter 3

Technical Background & Preliminaries

General Notation & Definitions

With respect to an arbitrary set X, we adopt the following notations:

- the power set of X is 2^X;
- the cardinality of X is $|X|$;
- the set of all probability distributions over X is $\mathrm{Dist}(X)$, where $P(x)$ is the probability of $x \in X$ with respect to $P \in \mathrm{Dist}(X)$,
- the identity function $\mathrm{Id}_X : X \to X$ maps every element to itself.

Let $\emptyset$ denote the empty set.

Numbers

We set the following notation for common number sets:

- $\mathbb{N} = \{0, 1, 2, ...\}$ is the set of *natural numbers*;
- $\mathbb{Z} = \{..., -2, -1, 0, 1, 2, ...\}$ is the set of *integers*;
- $\mathbb{Q} = \left\{\frac{n}{d} \ : \ n, d \in \mathbb{Z}\right\}$ is the set of *rational numbers*;
- $\mathbb{R}$ is the set of *real numbers*.

Vectors & Matrices

Denote the set of all $m \times n$ dimensional matrices of real numbers by $\mathbb{R}^{m\times n}$. We identify the set of matrices $\mathbb{R}^{m\times 1}$ with the set of vectors $\mathbb{R}^m$ and the set of matrices $\mathbb{R}^{1\times n}$ with the set of n-dimensional row vectors. By default, a vector is assumed to be a column vector. For a vector $\boldsymbol{v} \in \mathbb{R}^m$ and an index $i \in \{1,...,m\}$, let $\boldsymbol{v}_i$ denote its i^{th} entry. Let $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ be an arbitrary matrix. For indices $i \in \{1,...,m\}$ and $j \in \{1,...,n\}$, we write $\boldsymbol{M}[i,\cdot] \in \mathbb{R}^{1\times n}$ and $\boldsymbol{M}[\cdot,j] \in \mathbb{R}^{m\times 1}$, respectively, for the i^{th} row vector and the j^{th} column vector of $\boldsymbol{M}$. Similarly, we write $\boldsymbol{M}[i,j]$ for the entry of $\boldsymbol{M}$ at row i and column j. We write $\mathbf{1}_n$ for the vector of size n with all entries equal to 1. and I_n for the identity matrix of dimension n. The dot product two vectors $\boldsymbol{u},\boldsymbol{v} \in \mathbb{R}^m$ is defined as $\boldsymbol{u}\boldsymbol{v} = \sum_{i=1}^m \boldsymbol{u}_i\boldsymbol{v}_i$. The transpose $\boldsymbol{M}^\intercal \in \mathbb{R}^{n\times m}$ of matrix $\boldsymbol{M}$ is obtained by reflecting $\boldsymbol{M}$ over its main diagonal so that $\boldsymbol{M}^\intercal[i,j] = \boldsymbol{M}[j,i]$. For two matrices $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{N} \in \mathbb{R}^{n\times p}$, their product $\boldsymbol{M}\boldsymbol{N} \in \mathbb{R}^{m\times p}$ is defined in the standard way so that $(\boldsymbol{M}\boldsymbol{N})[i,j] = \boldsymbol{M}^\intercal[i,\cdot]\boldsymbol{N}[\cdot,j]$.

A *linear function* $f : \mathbb{R}^n \to \mathbb{R}^m$ is one for which the equations $f(\boldsymbol{u}) + f(\boldsymbol{v}) = f(\boldsymbol{u}+\boldsymbol{v})$ and $f(a\boldsymbol{v}) = af(\boldsymbol{v})$ hold for all scalars $a \in \mathbb{R}$ and vectors $\boldsymbol{u},\boldsymbol{v} \in \mathbb{R}^n$. Any linear function $f : \mathbb{R}^n \to \mathbb{R}^m$ can be represented as a matrix $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ such that $f(\boldsymbol{v}) = \boldsymbol{M}\boldsymbol{v}$ holds for all $\boldsymbol{v} \in \mathbb{R}^n$. An *affine function* $f : \mathbb{R}^n \to \mathbb{R}^m$ is the sum of a linear function and a constant so that $f(\boldsymbol{v}) = \boldsymbol{M}\boldsymbol{v} + \boldsymbol{b}$ for some matrix $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ and vector $\boldsymbol{b} \in \mathbb{R}^m$.

Let $\boldsymbol{x} \in \mathbb{R}^n$ be a vector of variables. A *linear constraint* of dimension n is an inequality of the form $\boldsymbol{a}\boldsymbol{x} \leq b$, where $\boldsymbol{a} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. A *linear constraint system* (LCS) is a conjunction of linear constraints, taking the form $\boldsymbol{M}\boldsymbol{x} \leq \boldsymbol{b}$ for some $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$.

3.1 Formal Languages & Automata

An *alphabet* Σ is a finite set of symbols. *Concatenating* two symbols $\sigma_1,\sigma_2 \in \Sigma$ produces a *word* or *string*, which is expressed by the juxtaposition $\sigma_1\sigma_2$ (or, as concatenation is not commutative, $\sigma_2\sigma_1$). In general, a word w over Σ is a concatenation $\sigma_1\sigma_2...\sigma_n$ of finitely many symbols, all which are elements of Σ. When there is no risk of confusion between concatenation and numerical multiplication, we use product notation to denote iterated concatenation, so that $\prod_{k=1}^n w_k = w_1...w_n$. The length of a word w,

denoted by $|w|$, is the number of constituent symbols making it up. For $\sigma \in \Sigma$ and $w \in \Sigma^*$, let $\sigma \in w$ indicate that the symbol σ occurs in the string w and let $|w|_\sigma$ be the number of occurrences of σ within w. The empty word, of length 0, is denoted by ε. We write Σ^n for the set of all words of length n and $\Sigma^{\leq n}$ for the set $\bigcup_{k=0}^{n} \Sigma^k$ of all strings of length at most n. The set of all finite words over Σ is defined as $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$. A subset $L \subseteq \Sigma^*$ is a called a *language*. Concatenation is extended to languages so that $L_1 L_2 = \{w_1 w_2 \,:\, w_1 \in L_1 \text{ and } w_2 \in L_2\}$, for $L_1, L_2 \subseteq \Sigma^*$.

An *ω-string* w over Σ is an infinite sequence $w = \sigma_0 \sigma_1 \ldots$ of symbols from Σ. We write Σ^ω for the set of all ω-strings over Σ. An *ω-language* L over Σ is a subset of Σ^ω.

A *morphism*, over alphabets Σ and Γ, is a function $h : \Sigma^* \to \Gamma^*$ which respects concatenation such that $h(\varepsilon) = \varepsilon$ and $h(w_1 w_2) = h(w_1)h(w_2)$, for all $w_1, w_2 \in \Sigma^*$. Every morphism is completely specified by its restriction to Σ. With respect to $X \subseteq \Sigma$, let $\mathrm{Er}_X : \Sigma^* \to (\Sigma \smallsetminus X)^*$ be the *erasing morphism*, defined such that $\mathrm{Er}_X(x) = \varepsilon$ for all $x \in X$ and $\mathrm{Er}_X(a) = \sigma$ for each $\sigma \in \Sigma \smallsetminus X$. A morphism $h : \Sigma^* \to \Gamma^*$ is *non-erasing* if $h(\sigma) \neq \varepsilon$ holds for every symbol $\sigma \in \Sigma$; if $h(\sigma) \in \Gamma$ holds for each $\sigma \in \Sigma$, then h is *letter-to-letter*.

Definition 3.1 – Nondeterministic Finite-State Automaton (NFA)

Syntax: A nondeterministic finite-state automaton $\mathcal{A}$ is specified by a tuple $\langle \Sigma, Q, q_0, F, \delta \rangle$, where

- Σ is a finite alphabet;
- Q is a finite set of states;
- $q_0 \in Q$ is a distinguished initial state;
- $F \subseteq Q$ is a set of accepting or final states;
- $\delta : Q \times \Sigma \to 2^Q$ is a transition function.

Semantics: Define an extended transition function $\delta^* : Q \times \Sigma^* \to 2^Q$ such that $\delta^*(q, \varepsilon) = q$ and $\delta^*(q, \sigma w) = \bigcup_{q' \in \delta(q,\sigma)} \delta^*(q', w)$, for any state $q \in Q$, symbol $\sigma \in \Sigma$, and word $w \in \Sigma^*$. The language $L(\mathcal{A}) \subseteq \Sigma^*$ recognized by $\mathcal{A}$ defined as

$$L(\mathcal{A}) = \{w \,:\, \delta^*(q_0, w) \cap F \neq \emptyset\}.$$

An NFA is *deterministic* (DFA) if $|\delta(q, \sigma)| = 1$, for every state q and every symbol σ. In this case, we write the type of the transition function as $Q \times \Sigma \to Q$. For every NFA, there is a DFA that recognizes the same language. A language is *regular* if, and only if, it is the language of some DFA.

For further details on formal languages and automata, we refer the reader to Hopcroft, Motwani,

and Ullman [HMU07], Lawson [Law04], Sipser [Sip97], and Sakarovitch [Sak09].

3.2 Markov Chains

A *Markov chain* M is a stochastic process given as a pair (X, P), where

- X is a set of states;
- $P : X \to \mathrm{Dist}(X)$ is a stochastic transition function that determines the probabilities of moving between various states[1] .

A *path* in a Markov chain is a finite sequence of states, and a *trajectory* is an infinite sequence of states. We adopt some notational conventions from formal languages, treating paths as finite strings over X and trajectories as ω-strings over X. Let $\mathrm{Path}(M) \subseteq X^*$ and $\mathrm{Traj}(M) \subseteq X^\omega$ denote, respectively, the sets of all paths and all trajectories over a Markov chain M. Further, for $n \in \mathbb{N}$, let $\mathrm{Path}_n(M)$ denote the set of all paths of length n, and let $\mathrm{Traj}_x(M)$ be the set of all trajectories having initial state $x_0 = x$. When the Markov chain in question is clear from context, we omit the M from this notation and write Path, Traj_x, *etc.* The length $|\mathfrak{p}|$ is the number of elements in the path $\mathfrak{p}$. Given two paths $\mathfrak{p} = x_1 \ldots x_n$ and $\mathfrak{q} = y_1 \ldots y_m$, we use concatenation to represent conjoined paths $\mathfrak{pq} = x_1 \ldots x_n y_1 \ldots y_m$ and $\mathfrak{qp} = y_1 \ldots y_m x_1 \ldots x_m$. If $\mathfrak{t} = y_1 y_2 \ldots$ is a trajectory, then prepending a path $\mathfrak{p}$ produces a new trajectory $\mathfrak{pt} = x_1 \ldots x_n y_1 y_2 \ldots$ To each path $\mathfrak{p} = x_1 \ldots x_n$, there is an associated probability given by the product $\prod_{k=1}^{n-1} P(x_{k+1} \mid x_k)$. For $n \in \mathbb{N}$ and states $x, y \in X$, let the probability of reaching y from x in n transitions be defined as the sum of probabilities for each length n path starting at x and ending at y: $P^n(y \mid x) = \sum_{\mathfrak{p} \in \mathrm{Path}_n} \prod_{k=1}^{n-1} P(x_{k+1} \mid x_k)$. There exists a unique probability measure $\mathbb{P}_M : \mathrm{Traj} \to [0, 1]$ that extends the path measures to the domain of trajectories. For trajectory $\mathfrak{t} \in \mathrm{Traj}$ and indices $n, m \in \mathbb{N}$ such that $n \leq m$, let

$$\mathfrak{t}_{[n,m]} = x_n x_{n+1} \ldots x_{m-1} x_m \qquad \text{and} \qquad \mathfrak{t}_{[m,n]} = x_m x_{m-1} \ldots x_{n+1} x_n .$$

Given $\mathfrak{t} \in \mathrm{Traj}$, define a subset of Path, comprising all paths occurring infinitely often in $\mathfrak{t}$, as

$$\mathrm{Inf}(\mathfrak{t}) = \{\mathfrak{p} \in \mathrm{Path} \ : \ \forall n.\, \exists m \geq n.\, \mathfrak{p} = \mathfrak{t}_{[m, m+|\mathfrak{p}|]}\}.$$

[1] We write $P(y \mid x)$ in place of $P(x)(y)$ for the probability of moving to state y from state x.

If there is a path from state x to state y, then y is said to be *accessible* from x. Two states x and y that are mutually accessible are called *communicating*. The communication relation is an equivalence relation that partitions X into subsets called *communicating classes*, each of which consists of mutually accessible states. A communicating class C is *closed* if, for every $x \in C$ and $y \in X \setminus C$, state y is not accessible from x. The *period* of a state x is defined as $\gcd\{n \in \mathbb{N} : 0 < P^n(x \mid x)\}$. A Markov chain is

- *irreducible* if it has a single communicating class;
- *aperiodic* if each state has period 1;
- *ergodic* if it is irreducible and aperiodic.

For further reading on Markov chains, we refer the reader to Norris [Nor98] and Ching et al. [Chi+13]

3.3 Decision Processes

The most general environments for sequential optimization are non-Markov decision processes.

Definition 3.2 – Non-Markov Decision Process (NMDP)

A non-Markov decision process is given by a tuple $\langle X, A, P, R \rangle$ in which

- X is a set of states;
- A is a set of actions;
- $P : X^* \times A \to \mathrm{Dist}(X)$ is a history-dependent probabilistic transition function;[a]
- $R : X \times^* \times A \to \mathrm{Dist}(\mathbb{R})$ is a history-dependent probabilistic reward function*a*.

[a] We write $P(x \mid \mathfrak{p}, a)$ and $R(r \mid \mathfrak{p})$ in place of $P(\mathfrak{p}, a)(x)$ and $R(\mathfrak{p})(r)$, respectively.

A Markov decision process (MDP) is a decision process with Markovian dynamics and a deterministic reward function. MDPs may be thought of as a controllable generalization of Markov chains.

Definition 3.3 – Markov Decision Process (MDP)

Formally, a Markov decision process M is given by a tuple $\langle X, A, P, R\rangle$, where

- X is a set of states;
- A is a set of actions;
- $P : X \times A \to \mathrm{Dist}(X)$ is a probabilistic Markovian transition function[a];
- $R : X \times A \to \mathbb{R}$ is a deterministic Markovian reward function.

[a] We write $P(y \mid x, a)$ in place of $P(x, a)(y)$.

Remark 3.1

Note that some parts of this manuscript use slightly different definitions of MDPs.

- In Chapters 4 and 5 each MDPs is defined to include a distinguished starting state x_0.
- In Chapter 4, we consider labeled MDPs with reward functions defined over these labels rather than directly over the states/actions.
- In Chapter 8, we consider MDPs that have reward functions of type $R : X \times X \to \mathbb{R}$, which map each state-transition to a reward, rather than each state-action pair.

A path in an MDP is a finite sequence of states in X^* such that $0 < \sum_{a \in A} P(x_{k+1} \mid x_k, a)$ holds at every index k. A trajectory in an MDP is an infinite sequence in X^ω satisfying the same constraint. We reuse the Path and Traj notations defined above for Markov chains for MDPs as well.

Behavioral Policies

A *policy* over an MDP M is a function π : Path $\to$ Dist(A) that returns distributions over the action set, given a path and current state[2] . Let Π_M be the set of all policies over M. Fixing a policy π on M induces a (generally non-Markov) stochastic process $M_\pi = \langle X, P_\pi\rangle$ with transition function P_π : Path $\to$ Dist(X) defined by

$$P_\pi(y \mid \mathfrak{p}) = \sum_{a \in A} P\big(y \mid x_{|\mathfrak{p}|}\big)\pi(a \mid \mathfrak{p}).$$

In this situation, the probability of a given path $\mathfrak{p}$ is $\prod_{k=1}^{|\mathfrak{p}|} P_\pi(x_k \mid \mathfrak{p}_{[1,k]})$. These path measures can be extended to obtain a unique, for each state x, distribution $\mathbb{P}_M^{\pi,x}$ on trajectories.

A policy π is

- *deterministic* if, for every path $\mathfrak{p}$, there exists an action a such that $\pi(a \mid \mathfrak{p}) = 1$;

[2] We write $\pi(a \mid \mathfrak{p})$ to denote the probability of action a according to π, given path $\mathfrak{p}$. When referring to the distribution over actions without referencing any particular probabilities, we write $\pi(\mathfrak{p})$.

- *stationary* if the equivalence of distributions $\pi(\mathfrak{p}) = \pi\big(x_{|\mathfrak{p}|}\big)$ holds for every path $\mathfrak{p}$;
- *positional* if it is both stationary and deterministic;
- *finite-memory* if the distribution $\pi(\mathfrak{p})$ can be computed using $O(1)$ space with respect to $|\mathfrak{p}|$;
- *unbounded-memory* if there exists a path $\mathfrak{p}$ such that $\omega(1)$ space[3] , with respect to the length of $\mathfrak{p}$, is required to compute the distribution $\pi(\mathfrak{p})$.

Let Π_M, Π_M^S, Π_M^D, and Π_M^P be, respectively, the sets of all strategies, stationary strategies, deterministic strategies, and positional strategies over M.

Objectives & Optimality

An *objective function* $\mathsf{F} : \mathbb{R}^\omega \to \mathbb{R}$, also known as a *payoff function*, maps infinite sequences of real numbers to real scalars. Some well known relevant payoff functions are defined, with respect to the reward sequence $\mathfrak{r} = (r_n)_{n\in\mathbb{N}} \in \mathbb{R}^\omega$:

$$\mathsf{I}(\mathfrak{r}) = \inf_{n\geq 0} r_n, \qquad \text{(Infimum Payoff)}$$

$$\mathsf{S}(\mathfrak{r}) = \sup_{n\geq 0} r_n, \qquad \text{(Supremum Payoff)}$$

$$\mathsf{L}^\flat(\mathfrak{r}) = \liminf_{n\to\infty} r_n, \qquad \text{(Lower Limit Payoff)}$$

$$\mathsf{L}^\sharp(\mathfrak{r}) = \limsup_{n\to\infty} r_n, \qquad \text{(Upper Limit Payoff)}$$

$$\mathsf{D}_\lambda(\mathfrak{r}) = \lim_{n\to\infty} \sum_{k=1}^{n} r_k \lambda^{k-1}, \qquad (\lambda\text{-Discounted Payoff, } \lambda \in [0,1))$$

$$\mathsf{A}(\mathfrak{r}) = \liminf_{n\to\infty} \sum_{k=1}^{n} \frac{r_k}{n}. \qquad \text{(Limit-Average Payoff[4])}$$

Remark 3.2

While discounted and average payoffs can be found in any of the standard references [FV96; FS01; Put94] on sequential optimization, the other payoffs listed above are not as commonly studied. For further details around these objectives, we point the interested reader to the references [CDH09; CH07; CH08; Gim07].

[3] The "little omega" notation used here denotes asymptotic dominance: $f(n) = \omega(g(n))$ means that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

Objective functions provide a means of quantitatively evaluating policies over an MDP M. Fixing an objective function F, a policy π, and an initial state x, define $\mathbb{E}_M^{\pi,x}[\mathsf{F}]$, with respect to the distribution $\mathbb{P}_M^{\pi,x}$, as the expectation of F taken over all trajectories generated by π starting from x. A policy is *optimal* for the objective F if it maximizes the expected value of F. This maximal expectation is called the F-*value* and is defined formally by the equation

$$\mathrm{Val}_M(\mathsf{F}, x) = \sup_{\pi \in \Pi_M} \mathbb{E}_M^{\pi,x}[\mathsf{F}].$$

The F-value is *computable* iff there is an algorithm that returns $\mathrm{Val}_M(\mathsf{F}, x)$ when given an MDP M and initial state x as input. The F-value is *approximable* iff there is an algorithm that returns a value V such that $|\mathrm{Val}_M(\mathsf{F}, x) - V| \leq \epsilon$, when given an MDP M, an initial state x, and a tolerance $\epsilon > 0$ as input. Given an MDP M, an initial state x, and a lower bound b, the F-*value problem* asks to decide whether the inequality $b \leq \mathrm{Val}_M(\mathsf{F}, x)$ holds.

For the values of discounted and limit-average payoffs, we typically use the shorthand notations

$$V_\lambda(x) = \mathrm{Val}_M(\mathsf{D}_\lambda, x) \qquad \text{and} \qquad V(x) = \mathrm{Val}_M(\mathsf{A}, x)$$

when M is clear from the context.

For further information on decision processes, we refer the reader to Filar and Vrieze [FV96], Feinberg and Shwartz [FS01], and Puterman [Put94].

3.4 Sequential Optimization Problems & Algorithms

We divide sequential optimization problems into two categories: *planning problems*, where agents have access to the specification of the environment, and *learning problems*, where the agent starts out with no knowledge about its environment. In the following two subsections, let the environment be an MDP $M = \langle X, A, P, R \rangle$.

3.4.1 Planning

Planning describes the situation in which the goal is to compute either the value or an optimal policy over some environment—assumed to be finite for computational purposes—that the agent

has full knowledge of, *i.e.* a complete description of the MDP is available. Typically, the objective under consideration is the discounted payoff, but optimization under long-run average payoffs has also received significant attention.

The basis of the dynamic programming algorithm known as *value iteration* for computing discounted values[5] over finite MDPs is an alternative characterization of the discounted value as the unique solution to the so-called *optimality equation* or *Bellman equation*

$$V_\lambda(x) = \max_{a \in A} R(x,a) + \lambda \sum_{y \in X} P(y \mid x,a) V_\lambda(y). \qquad \text{(Bellman Optimality Equation)}$$

As there are many variants of value iteration that are tailored to specific scenarios, value iteration forms one of the main avenues of the dynamic programming approach to sequential optimization. The other main category of dynamic programming algorithms are based on a scheme called *policy iteration*. A third approach, which is not based on dynamic programming, reduces the problems of computing values and optimal policies to linear programming problems. All three of these methods can be executed in polynomial time.

For further information regarding planning problems and algorithms over MDPs, we refer the reader to Filar and Vrieze [FV96], Feinberg and Shwartz [FS01], and Puterman [Put94].

3.4.2 Reinforcement Learning

Reinforcement learning (RL) [SB98] is a sampling-based method for sequential optimization in uncertain environments, through episodic interactions. Each episode consists of a sequence of experiences in which the agent chooses an action from a given state, observes the resulting effect, and receives an associated reward. Choosing a particular action at a particular state determines a probability distribution over the environment's states, and the state transitions stochastically based on this distribution. A sequence of states and actions is generated by the repeated interaction of an agent with its environment, and this sequence is mapped, via a reward function, to a sequence of scalar values called a reward signal. An RL problem instance places an agent into an unknown environment

[5] Value iteration algorithms have also been developed for a number of other payoffs [CH08].

and tasks it with optimizing the value of an objective function that aggregates reward sequences in some fashion. A solution to an RL problem is a behavioral policy or strategy that optimizes the overall payoff in expectation. An RL algorithm is a method by which an optimal policy may be constructed mechanically via repeated interaction with the environment. An RL algorithm is said to converge if the policy maintained by an agent following its instructions tends asymptotically towards optimality. The existence of convergent RL algorithms is provably guaranteed only for finite environments, where agents can maintain a policy in the form of a finite table.

The standard RL scenario assumes a discounted objective, and model-free approaches typically leverage the *state-action value* or *Q-value*: defined as the optimal value from state x, given that action a has been selected, and is the solution of the equation

$$Q_\lambda(x,a) = R(x,a) + \lambda \sum_{y\in X} P(y \mid x,a) \max_{b\in A} Q_\lambda(y,b). \qquad \text{(State-Action Value)}$$

Notice that $V_\lambda(x) = \max_{a\in A} Q_\lambda(x,a)$.

The Q-value provides the foundation for the classic Q-Learning algorithm, which learns an optimal policy by approximating Q_λ with a sequence Q_n of maps which asymptotically converge to Q_λ. In particular, Q_1 is initialized arbitrarily. Then, at each time n, the agent selects an action a_n from the current state x_n, receives reward r_n, observes the next state x_{n+1}, and performs the update

$$Q_{n+1}(x_n,a_n) := (1-\alpha_n)Q_n(x_n,a_n) + \alpha_n\Big(r_n + \lambda \max_{a\in A} Q_n(x_{n+1},a)\Big) \tag{3.1}$$

in which y is the next state as determined by the outcome of sampling the distribution $P(x,a)$ and the numbers $\alpha_n \in (0,1)$ are time-dependent parameters called *learning rates*.

An important result in RL gives a sufficient condition for asymptotic convergence of the Q-learning algorithm.

Theorem 3.1 – Watkins and Dayan [WD92]

If every state-action pair in a finite MDP is encountered infinitely often and learning rates satisfy the Robbins-Monro conditions [RM51] $\sum_{n=1}^{\infty} \alpha_n = \infty$ and $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$, then Q-learning, based on iterating the update (3.1), converges almost surely in the limit:

$$Q_n \xrightarrow{\text{a.s.}} Q_\lambda \qquad \text{as} \qquad n \to \infty.$$

Deep Reinforcement Learning

Deep RL [Pla22] describes an approximate form of RL where the finite map representation of policies used in tabular RL is replaced by an artificial neural network. In spite of a lack of theoretical guarantees around aspects like convergence, deep RL has been fruitfully applied in environments with infinite state spaces. In this setting, an RL algorithm is a method by which an agent's interactions are used to train a neural network to approximate an optimal policy.

Part I

Modeling Rewards & Environments With Regular Transformations

Chapter 4

Probabilistic Reward Machines In Reinforcement Learning

Traditionally, reinforcement learning (RL) has relied on strong assumptions related to Markovian dynamics of underlying environments. In particular, the reward signal through which an agent receives positive or negative reinforcement is typically defined in terms of a Markovian reward function $R : X \times A \to \mathbb{R}$ of the current state of the environment and the action of an agent. Histories of observations are not considered in such settings. However, many problem domains require taking history into account. Examples include learning with sparse rewards [Nei+21], learning when rewards are defined in terms of regular expressions & formal logics [Cam+19], and learning under conditions of partial observability [Ica+19].

Recently, the problem of learning structured representations of non-Markovian reward signals has received significant attention. These representations often take the form of finite state machines called *reward machines* [Ica+18; Ica+22]. Reward machines can be embedded into the state space of decision processes to yield larger Markovian decision processes, thereby facilitating the use of classical techniques for MDPs in situations with non-Markovian reward dynamics [GB20; Xu+21]. They also serve as a memory mechanism for reasoning over partially observable environments [Ica+19], are useful for reward shaping to mitigate sparse reward signals [Cam+19; VM20; Vel+21], and provide explanations of RL systems [Xu+21].

While much progress has been made on learning and leveraging reward machines [Xu+20; Xu+21; AB20; GB20; Nei+21; Ren+21], the more general setting where rewards exhibit both non-Markovian and stochastic dynamics has not been addressed. We make progress on this front by introducing *probabilistic*

reward machines (PRMs). We present an algorithm to learn PRM representations of non-Markovian stochastic reward signals by combining RL with a probabilistic variant of the $L^\star$ algorithm [Ang87] for automaton inference. Our algorithm, called $RL^\star$, uses a product construction to produce a standard MDP from the environmental decision process and the learned PRMs in order to perform RL, and we provide an accompanying proof that these products correctly simulate the original decision processes with non-Markovian stochastic rewards. To further motivate PRMs, we show that an exponential blowup in occurs when embedding probabilities in the environment and using a deterministic reward machine. Lastly, we establish theoretical guarantees regarding the asymptotic convergence of $RL^\star$.

Remark 4.1

This chapter is based on the ICAPS 2022 paper *Inferring Probabilistic Reward Machines from Non-Markovian Reward Signals for Reinforcement Learning* [Doh+22]. Independently, the AAAI 2022 paper *Reinforcement Learning with Stochastic Reward Machines* [CGN22] was published at nearly the same time. Despite the close similarity between these two publications, neither group of authors was aware of the others' work until both papers had appeared in the proceedings of their respective conferences.

Transition-Markov Decision Processes

In this chapter, the environment is not an MDP, but a *transition-Markov decision process* (TMDP). A TMDP maintains the classical assumption that state transitions are Markovian while generalizing the standard MDP model by allowing history-dependent stochastic rewards.

Definition 4.1 – Transition-Markov Decision Process (TMDP)

Syntax: A transition-Markov decision process is given by a tuple $\langle X, x_I, A, P, \Lambda, \ell, R\rangle$, where

- X is a finite set of states;
- $x_I \in X$ is an initial state;
- A is a finite set of actions;
- $P : X \times A \to \text{Dist}(X)$ is a probabilistic transition function;
- Λ is a finite alphabet of labels[a];
- $\ell : X \times A \times X \to \Lambda$ is a function assigning a label to each transition;
- $R : \Lambda^* \to \text{Dist}(\mathbb{R})$ is a non-Markovian stochastic reward function.

Semantics: The semantics of a T take the form of a function $[\![T]\!] : A^* \to \text{Dist}(\Lambda^* \times \mathbb{R})$ such that

$$[\![T]\!](w)(l,r) = \begin{cases} 1 & \text{if } w = l = \varepsilon \text{ and } r = 0, \\ 0 & \text{if } w, l \text{ is not observable,} \\ p\, R(r \mid l) & \text{if } w, l \text{ is observable with probability } p. \end{cases}$$

[a] The labeling alphabet Λ is interpreted as the powerset 2^{AP} of some finite set AP of atomic propositions.

Additionally, define the language of T as the set

$$L(T) = \{\langle w, l\rangle \in A^* \times \Lambda^* : \exists r \in \mathbb{R}.\ [\![T]\!](w)(l,r) \neq 0\}.$$

For an MDP M and a positive integer n,the subset of strings from $L(M)$ that are at most length n is defined as $L^{\leq n}(M) = L(M) \cap \bigcup_{k \leq n} \left(A^k \times \Lambda^k\right)^{\leq n}$.

A run of a TMDP is a string $x_0 a_1 x_1 ... a_n x_n \in X(AX)^*$ such that $0 < \prod_{k=1}^{n} P(x_k \mid x_{k-1}, a_k)$. For a run $x_0 a_1 x_1 ... a_n x_n$, its corresponding label sequence is $\lambda_1 \lambda_2 ... \lambda_n$, where $\lambda_k = \ell(x_{k-1}, a_k, x_k)$, and the corresponding reward sequence is $r_1 r_2 ... r_n$ where r_k is determined according to the distribution $R(\lambda_1 ... \lambda_k)$. A trajectory is a run for which $x_0 = x_I$.

4.1 Probabilistic Reward Machines

We now define the *probabilistic reward machine* model.

Definition 4.2 – Probabilistic Reward Machine (PRM)

A probabilistic reward machine is given by a tuple $\langle \Lambda, \Gamma, Q, q_I, \delta, \rho \rangle$, where

- Λ is the input alphabet of labels;
- $\Gamma \subset \mathbb{R}$ is a finite set of rewards;
- Q is a finite set of states;
- $q_I \in Q$ is a distinguished initial state;
- $\delta : Q \times \Lambda \to \text{Dist}(Q)$ is a probabilistic transition function;
- $\rho : Q \times \Lambda \times Q \to \Gamma$ is a function mapping each transition to a reward from Γ.

A run of a PRM H is a sequence of transitions $q_0 \xrightarrow[\lambda_1]{\gamma_1} q_1 \ldots \xrightarrow[\lambda_n]{\gamma_n} q_n$ where $\gamma_k = \rho(q_{k-1}, \lambda_k, q_k)$ and $\delta(q_k \mid q_{k-1}, \lambda_k) > 0$ hold at each k. Let $\boldsymbol{H} : \Lambda\Gamma \to [0,1]^{|Q|\times|Q|}$ map each pair $\lambda, \gamma \in \Lambda \times \Gamma$ to a transition matrix $\boldsymbol{H}(\lambda\gamma)$ such that

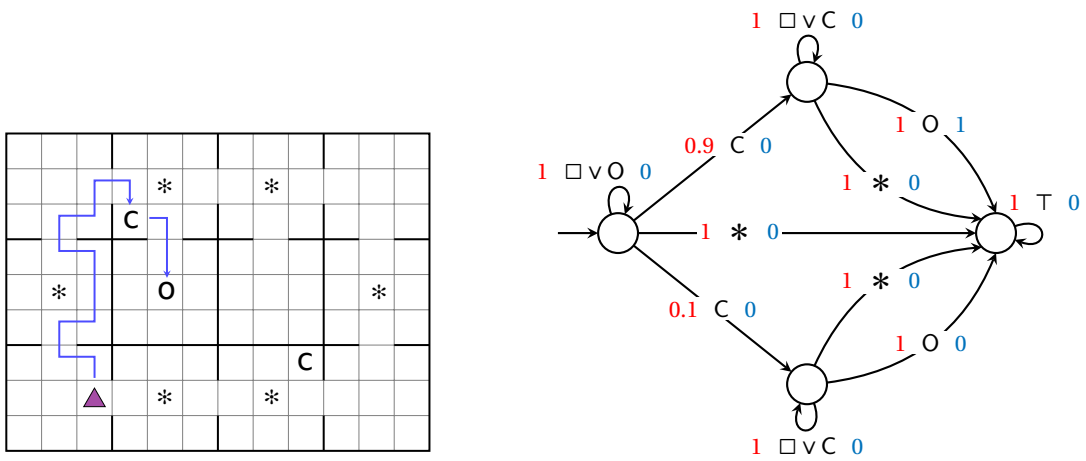
$$\boldsymbol{H}(\lambda\gamma)[i,j] = \begin{cases} \delta\left(q_j \mid q_i, \lambda\right) & \text{if } \rho\left(q_i, \lambda, q_j\right) = \gamma, \\ 0 & \text{otherwise.} \end{cases}$$

As usual, we extend the domain from $\Lambda\Gamma$ to $(\Lambda\Gamma)^*$ by setting $\boldsymbol{H}(\lambda\gamma w) = \boldsymbol{H}(\lambda\gamma)\boldsymbol{H}(w)$ and $\boldsymbol{H}(\varepsilon) = \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix and ε the empty string. A distribution $\mathbb{P}_H : \Lambda^* \times \Gamma^* \to [0,1]$ over pairs of inputs and reward sequences is induced:

$$\mathbb{P}_H(\lambda_1 \ldots \lambda_n, r_1 \ldots r_n) = \boldsymbol{q_I}\boldsymbol{H}(\lambda_1 r_1 \ldots \lambda_n r_n)\mathbf{1},$$

where $\boldsymbol{q_I}$ is an initial distribution over Q (typically with an entry of 1 at the index for state q_I and zeros everywhere else) and $\mathbf{1}$ is the vector with 1 at each entry. The PRM H is said to encode the reward function R of a TMDP if, for any label sequence $l = \lambda_1 \ldots \lambda_n$ and reward sequence $r = r_1 \ldots r_n$, the following equality holds:

$$\mathbb{P}_H(l, r) = \prod_{k=1}^{n} R(r_k \mid \lambda_1 \ldots \lambda_k).$$



(a) Office Grid-World.

(b) A PRM for the office grid-world.

Figure 4.1: The office grid-world and a PRM encoding the reward signal from Example 4.1.

Example 4.1 – Office Grid-World

Consider the office grid-world environment of [Ica+18], as displayed in . The original task, which takes the form of a deterministic reward machine, requires that the agent acquire coffee at location c and delivers the coffee to office o, at which point the agent receives a reward of 1. If the agent steps on a $*$, they fail the task and observe a reward of 0. A variant of this task, in which the reward signal is probabilistic, may be obtained by introducing a 10% chance that the coffee machine malfunctions, producing weak coffee which is rejected upon delivery and receives a reward of 0. A graphical representation of a PRM for this task is shown in Figure 4.1b. Each edge is marked by a probability, a label, and a reward.

The following property, which we call *reward-determinism* , will prove significant in learning PRMs.

Definition 4.3 – Reward-Determinism

A PRM is reward-deterministic if, for any $q, q_1, q_2 \in Q$ and $l \in \Lambda$ such that $\delta(q_1 \mid q, l) > 0$ and $\delta(q_2 \mid q, l) > 0$, it holds that either $q_1 = q_2$ or $\rho(q, l, q_1) \neq \rho(q, l, q_2)$.

In other words, a state and a label-reward pair uniquely determine a successor state in a reward-deterministic PRM. The reward-determinism property is inspired by the label-determinism property used for learning MDPs in [Tap+19; Tap+21].

Taking the product of a TMDP and PRM yields a labelled MDP. We defined labeled MDPs now, since they differ slightly from the definition given in Chapter 3.

Definition 4.4 – Labeled Markov Decision Processes (MDPs)

Syntax: A labeled MDP M is given by a tuple $\langle X, x_0, A, P, \Lambda, \ell, R\rangle$, where

- X is a finite set of states;
- x_0 is a distinguished initial state;
- A is a finite set of actions;
- $P : X \times A \to \text{Dist}(X)$ is a probabilistic transition function;
- Λ is a finite alphabet of labels;
- $\ell : X \times A \times X \to \Lambda$ is a labeling function over transitions;
- $R : \Lambda \to \mathbb{R}$ is a Markovian reward function mapping each label to a real number.

Semantics: The semantics of a labeled MDP $M = \langle X, x_I, A, P, \Lambda, \ell, R\rangle$ is a function $[\![M]\!] : A^* \times \Lambda^* \to \text{Dist}(\mathbb{R})$ such that

$$[\![M]\!](w,l)(r) = \begin{cases} 1 & \text{if } w = \varepsilon \text{ and } l = \varepsilon \text{ and } r = 0 \\ 0 & \text{if } w, l \text{ is not observable} \\ p & \text{if } w, l \text{ is observable with probability } p \text{ and } l = l'\lambda \text{ and } r = R(\lambda) \end{cases}$$

Define the language of a MDP M as the set

$$L(M) = \{\langle w, l\rangle \in A^* \times \Lambda^* : \exists r \in \mathbb{R}.\ [\![M]\!](w,l)(r) \neq 0\}.$$

Two MDPs M, M' are equivalent if, and only if, the equation $[\![M]\!]\langle w, l\rangle = [\![M']\!]\langle w, l\rangle$ holds for any $w \in A^*$ and any $l \in \Lambda^*$.

The product operation between TMDPs and PRMS is defined as follows.

Definition 4.5 – Product: TMDP × PRM → MDP

Let $T = \langle X, x_I, A, P, \Lambda, \ell, R\rangle$ be a TMDP and let $H = \langle \Lambda, \Gamma, Q, q_I, \delta, \rho\rangle$ be a PRM. The product $T \otimes H$ is an MDP $\langle X', x'_I, A, P', \Lambda, \ell', R'\rangle$ such that

- $X' = X \times Q$;
- $x'_I = \langle x_I, q_I\rangle$;
- $P'\big(\langle x', q'\rangle \mid \langle x, q\rangle, a\big) = P\big(x' \mid x, a\big)\delta\big(q' \mid \ell\big(x, a, x'\big), q\big)$;
- $\ell'\big(\langle x, q\rangle, a, \langle x', q'\rangle\big) = \ell\big(x, a, x'\big)$;
- $R'\big(\ell'\big(\langle x, q\rangle, a, \langle x', q'\rangle\big)\big) = \rho\big(q, \ell\big(x, a, x'\big), q'\big)$.

The following theorem establishes that the product defined above is well-founded.

Theorem 4.1

If T is a TMDP with a reward encoded by a reward deterministic PRM H, then the equation

$$[\![T]\!](w)(l,\gamma) = [\![T \otimes H]\!](w,l)(\gamma)$$

holds for any $w \in A^*$, any $l \in \Lambda^*$, and any $\gamma \in \Gamma$.

Proof. Suppose that $T = \langle X, x_I, A, P, \Lambda, \ell, R\rangle$ is a TMDP, $H = \langle \Lambda, \Gamma, Q, q_I, \delta, \rho\rangle$ is a PRM encoding R, and $T \otimes H = \langle X', x_I', A, P', \Lambda, \ell', R'\rangle$ is their product constructed according to Definition 4.5. If $[\![T]\!](w)(l,\gamma) \neq 0$, then, by the semantics provided in Definition 4.1, there exists $p, q \in [0,1]$ such that w, l is observable with probability p and

$$[\![T]\!](w)(l,\gamma) = p\, R(\gamma \mid l).$$

If H encodes the reward function of T, then it follows from the definition of PRM semantics that

$$\mathbb{P}_H\left(l, \Gamma^{|l|-1}\gamma\right) = R(\gamma \mid l),$$

where $\Gamma^{|l|-1}$ denotes all reward sequences length $|l|-1$. Notice that the equation

$$\mathbb{P}_H(s, \Gamma^n) = \sum_{r \in \Gamma^n} \mathbb{P}_H(s,r) = 1$$

holds for any n and any $s \in \Lambda^*$. Suppose now that $l = l'\lambda$ and $w = w'a$. Assuming H is reward-deterministic, there must be states $q, q' \in Q$ such that

$$\mathbb{P}_H\left(l, \Gamma^{|l|-1}\gamma\right) = R(\gamma \mid l) = \delta\left(q' \mid q, \lambda\right) \quad \text{and} \quad \rho\left(q, \lambda, q'\right) = \gamma.$$

Moreover, the probability p of observing w, l in $T \otimes H$, depends only on the transition probability function P of T whenever the reward probabilities are not in consideration. Therefore, there exist states $x, x' \in X$ such that

$$\Pr[\langle w, l\rangle \in L(T)] = \Pr\left[\langle w', l'\rangle \in L(T \otimes H)\right] P\left(x' \mid x, a\right).$$

Since we have established that $R(\gamma \mid l) = \delta\big(q' \mid q, \lambda\big)$, we may conclude that

$$\begin{aligned}
[\![T]\!](w)(l,\gamma) &= \Pr[\langle w, l\rangle \in L(T)]\; R(\gamma \mid l) \\
&= \Pr\big[\langle w', l'\rangle \in L(T \otimes H)\big]\; P\big(x' \mid x, a\big)\, \delta\big(q' \mid q, \lambda\big) \\
&= \Pr[\langle w, l\rangle \in L(T \otimes H)] \\
&= [\![T \otimes H]\!](w, l)(\gamma). \qquad \square
\end{aligned}$$

The next result justifies the use of PRMs by establishing that they can provide a more compact representation of systems than might otherwise be modeled by embedding probabilities capturing the stochasticity of a stochastic non-Markovian reward function directly into the environment and using a deterministic reward machine to encapsulate the non-Markovian aspects of the reward function.

Theorem 4.2 – Succinctness Of Probabilistic Reward Machines

Suppose that T is a TMDP with reward function encoded by a reward-deterministic PRM H. If T has m states and H has n states, then there exists a TMDP T' with mn states with a deterministic reward function encoded by deterministic reward machine H' with $O(2^n)$ states such that the product MDPs $T \otimes H$ and $T' \otimes H'$ are equivalent.

Proof. Suppose that $T = \langle X, x_I, A, P, \Lambda, \ell, R\rangle$ is a TMDP, $H = \langle \Lambda, \Gamma, Q, q_I, \delta, \rho\rangle$ is a PRM encoding R. We proceed to construct a deterministic PRM as follows:

(1) We construct an NFA N from H such that rewards are moved into the states and a transition is included in N for every transition of positive probability in H.

(2) A DFA D is produced using a subset construction on N.

(3) A deterministic reward machine H' is obtained by interchanging deterministic transitions in D with transitions of probability 1 and bringing rewards back out onto the transitions while still keeping them in the states.

Note that reward-determinism of H ensures that the determinization step can be carried out without introducing sequences of rewards in H' that were unobtainable in H. The three steps above can be

composed into a single step to obtain a deterministic PRM $H' = \langle \Lambda, \Gamma, Q', q'_I, \delta', \rho' \rangle$, where

- $Q' = 2^{Q \times \Gamma}$;
- $q'_I = \{\langle q_I, 0 \rangle\}$;
- $\delta'(Y_2 \mid Y_1, \lambda) = \begin{cases} 1 & \text{if } \delta(q' \mid q, \lambda) > 0 \text{ and } \rho(q, \lambda, q') = \gamma', \text{ for all } \langle q, \gamma \rangle \in Y_1 \text{ and } \langle q', \gamma' \rangle \in Y_2 \\ 0 & \text{otherwise;} \end{cases}$
- $\rho'(Y_1, \lambda, Y_2) = \gamma$ iff $Y_2 \subseteq \{\langle q, \gamma \rangle : q \in Q\}$.

Next, we construct a TMDP $T' = \langle X', x'_I, A, P', \Lambda, \ell', R' \rangle$ as a modified product of H and T, where

- $X' = X \times Q$,
- $x'_I = \langle x_I, q_I \rangle$,
- $P'(\langle x', q' \rangle \mid \langle x, q \rangle, a) = P(x' \mid x, a)\, \delta(q' \mid q, \ell(x, a, x'))$,
- $\ell'(\langle x, q \rangle, a, \langle x', q' \rangle) = \ell(x, a, x')$, and
- $R'(l)(\gamma) = \mathbb{P}_{H'}\left(l, \Gamma^{|l|-1}\gamma\right)$.

The TMDP T' simulates the transition probabilities of H and has a deterministic reward function encoded by H'. Taking the product of T' and H' yields an MDP $T' \otimes H' = \langle X'', x''_I, A, P'', \Lambda, \ell'', R'' \rangle$ such that

- $X'' = X' \times Q'$,
- $x''_I = \langle x'_I, q'_I \rangle$,
- $P''(\langle x'_2, q'_2 \rangle \mid \langle x'_1, q'_1 \rangle, a) = P'(x'_2 \mid x'_1, a)\, \delta'(q'_2 \mid q'_1, \ell'(x'_1, a, x'_2))$,
- $\ell''(\langle x'_1, q'_1 \rangle, a, \langle x'_2, q'_2 \rangle) = \ell'(x'_1, a, x'_2)$, and
- $R''(\ell''(\langle x'_1, q'_1 \rangle, a, \langle x'_2, q'_2 \rangle)) = \rho'(q'_1, \ell'(x'_1, a, x'_2), q'_2)$.

What remains to be shown is that the product MDPs $T \otimes H$ and $T' \otimes H'$ are equivalent. We proceed by induction on $\langle w, l \rangle$.

Base case: $\langle w, l \rangle = \langle \varepsilon, \varepsilon \rangle$.

It follows immediately from Definition 4.4 that

$$[\![T \otimes H]\!](\varepsilon,\varepsilon)(\gamma) = [\![T' \otimes H']\!](\varepsilon,\varepsilon)(\gamma)$$

holds for all $\gamma \in \Gamma$.

INDUCTIVE CASE: *If* $[\![T \otimes H]\!](w', l') = [\![T' \otimes H']\!](w', l')$, *then* $[\![T \otimes H]\!](w'a, l'\lambda) = [\![T' \otimes H']\!](w'a, l'\lambda)$.

Let $w \in A^*$, $l \in \Lambda^*$, and $\gamma \in \Gamma$, and suppose that $w = w'a$ and $l = l'\lambda$. Then, the following derivation holds for some $x, x' \in X$, $q, q' \in Q$ and $Y_1, Y_2 \subseteq Q$.

$$\begin{aligned}
[\![T' \otimes H']\!](w,l)(\gamma) &= \Pr\big[\langle w,l\rangle \in L\big(T' \otimes H'\big) \text{ and } \gamma = R''(\lambda)\big] \\
&= \Pr\big[\langle w',l'\rangle \in L\big(T' \otimes H'\big)\big]\, P''\big(\langle\langle x',q'\rangle, Y_2\rangle \mid \langle\langle x,q\rangle, Y_1\rangle, a\big) \\
&= \Pr\big[\langle w',l'\rangle \in L\big(T' \otimes H'\big)\big]\, P\big(x' \mid x,a\big)\, \delta\big(q' \mid q,\lambda\big)\, \delta'(Y_2 \mid Y_1,\lambda) \\
&= \Pr\big[\langle w',l'\rangle \in L\big(T' \otimes H'\big)\big]\, P\big(x' \mid x,a\big)\, \delta\big(q' \mid q,\lambda\big) \\
&= [\![T' \otimes H']\!]\big(w',l'\big)(\gamma)\, P\big(x' \mid x,a\big)\, \delta\big(q' \mid q,\lambda\big) \\
&= [\![T \otimes H]\!]\big(w',l'\big)(\gamma)\, P\big(x' \mid x,a\big)\, \delta\big(q' \mid q,\lambda\big) \\
&= [\![T \otimes H]\!](w,l)(\gamma)
\end{aligned}$$

□

4.2 The RL* Algorithm

This section considers the RL setting where we wish to also learn a PRM representation of a stochastic non-Markovian reward signal.

Popular RL algorithms, such as Q-learning, are known to converge to the optimal policy only when the dynamics of an environment are perfectly observable (*i.e.* not obfuscated by noise or any other phenomenon) and Markovian. We make progress on this front by developing method for convergent RL in the presence of non-Markovian rewards with the assumption that reward signals are derived using an unknown finite probabilistic reward machine. Our algorithm facilitates explainability by simultaneously

inferring the structure of the unknown PRM through data produced by the agent's interaction with the environment. Our approach interleaves active automata learning algorithms (in the form of the $L^\star$ algorithm) with model-free reinforcement learning (in particular, Q-learning). This work builds upon the active MDP learning approach of [Tap+19; Tap+21] and the work of [Xu+21] on combining active learning with reinforcement learning for deterministic non-Markovian rewards.

Active Inference Of Deterministic Finite-State Automata

In the seminal work of Angluin [Ang87], an algorithm called $L^\star$ is proposed by which a DFA for a given regular language L can be learned through the use of a *minimally adequate teacher* that can answer *membership queries* and *equivalence queries*. When the learner executes a membership query, it presents a word $w \in \Sigma^*$ to the teacher and the teacher outputs whether $w \in L$. If the learner executes an equivalence query, it presents a hypothesis DFA to the teacher who must then answer whether this automaton encodes the language to be learned. If not, the teacher generates a counterexample in the form of a word on which the two languages differ.

The $L^\star$ algorithm maintains a data structure called the *observation table*.

Definition 4.6 – Observation Table

An observation table over an alphabet Σ is given by a tuple $\langle S, E, T \rangle$, where

- $S \subseteq \Sigma^*$ a prefix-closed set of *state words*;
- $E \subseteq \Sigma^*$ a suffix-closed set of *test words*;
- $T : (S \cup S\Sigma)E \to \{0,1\}$ is a mapping that stores whether various strings are elements of the target language.

$L^\star$ begins with initializing S and E as $\{\varepsilon\}$. As the algorithm proceeds, there are two critical properties, *closure* and *consistency*, that must be tracked. These properties are defined in terms of a notion called E-equivalence.

Algorithm 1: Pseudocode for the L* inference algorithm.

```
Input: Alphabet Σ
Initialize S and E to {ε};
Start filling observation table by issuing a membership query for each symbol σ ∈ Σ;
repeat
    while Observation table is open or inconsistent do
        Construct a string w that witnesses non-closure or inconsistency;
        Issue membership query for w and add the teacher's response to the observation table;
    Construct a DFA A from the closed and consistent observation table; Issue equivalence
     query for A;
    if Teacher answers with counterexample then
        Add counterexample to the observation table;
until Teacher answers equivalence query affirmatively;
return A;
```

Definition 4.7 – Closure & Consistency

Two words $w, w' \in \Sigma^*$ are E-equivalent, denoted $w \equiv_E w'$, if $T_{we} = T(w'e)$ holds for every $e \in E$. An observation table $\langle S, E, T \rangle$ is

CLOSED iff, for all $s, s' \in S$, the following implication holds: if $s \equiv_E s'$, then $s\sigma \equiv_E s'\sigma$ for all $\sigma \in \Sigma$;

CONSISTENT iff, for all $s \in S$ and $\sigma \in \Sigma$, there exists some $s' \in S$ such that $s\sigma \equiv_E s'$.

For a closed and consistent table, a corresponding automaton can be derived by taking each E-equivalence class of S to be a state, with the empty string ε as the starting state. The transition function is defined using the closure property. That is, whenever we have $s\sigma \equiv_E s'$, then we also have the transition $s \xrightarrow{\sigma} s'$ in the DFA. Furthermore, it follows from the consistency property that s' is unique, thereby implicitly enforcing determinism in the constructed automaton. The accepting states are determined by the strings in the table that are known to be in the language of the teacher.

The L* algorithm continuously works to ensure that the maintained observation table is closed and consistent, so that a DFA may be constructed to issue to the teacher via an equivalence query. If the teacher responds to an equivalence query with a counterexample, it is used to modify the observation table by adding the counterexample and all its prefixes to S. Membership queries are then used to determine the values of the new entries and ensure that the table becomes closed and consistent once more. A sketch of the algorithm is provided in Algorithm 1.

Active Inference Of Probabilistic Reward Machines

We now turn to the task of adapting the active inference approach of $L^\star$ for the purposes of PRM inference in the RL setting. In this setting, the main challenge arises from the fact that there is no oracle available to answer queries with certainty; queries must be answered indirectly through interactions with the environment.

The critical data structure employed by $RL^\star$ is a variation of Angluin's observation table that is tailored to hold frequency data obtained by sampling.

Definition 4.8 – Sampling Observation Table

For an underlying PRM $\langle \Lambda, \Gamma, Q, q_I, \delta, \rho \rangle$, an observation table is a tuple $\langle S, E, T \rangle$, where

- $S \subset (\Lambda\Gamma)^*$ is a prefix-closed set of *samples*;
- $E \subset (\Lambda\Gamma)^*\Lambda$ is a suffix-closed set of *experiments*;
- $T : S(SE \cup S\Lambda\Gamma E) \to (\Gamma \to \mathbb{N})$ is a map from label-reward words to reward frequencies[a].

[a] We often write the function $T_w : \Gamma \to \mathbb{N}$ as T_w.

Defining appropriate analogs of closure and consistency for this type of observation table relies on a notion of statistical difference, based on the Hoeffding bound.

Definition 4.9 – Statistical Difference

For a function $f : (\Lambda\Gamma)^*\Lambda \to (\Gamma \to \mathbb{N})$ and a word $w \in (\Lambda\Gamma)^*\Lambda$, let f_w denote the function $f(w) : \Gamma \to \mathbb{N}$. Two sequences $s, s' \in S\Lambda$ are *statistically different* with respect to f, written $\mathrm{Diff}_f(s, s')$, if

- either of $N = \sum_{g \in \Gamma} f_s(g)$ or $N' = \sum_{g \in \Gamma} f_{s'}(g)$ is zero,
- or there exists $\gamma \in \Gamma$ such that

$$\sqrt{\frac{1}{2}\ln\frac{2}{C}}\left(\sqrt{\frac{1}{N}} + \sqrt{\frac{1}{N'}}\right) < \left|\frac{f_s(\gamma)}{N} - \frac{f_{s'}(\gamma)}{N'}\right|,$$

where C is a data-dependent confidence level.[a]

[a] A good choice for C is m^{-3} where m is the number of samples taken thus far [Tap+19; Tap+21].

Two cells se and $s'e'$ of an observation table $\langle S, E, T \rangle$ are compatible, denoted $se \sim s'e'$, if $\mathrm{Diff}_T(se, s'e')$ is false. We say that two rows s, s' are compatible, denoted $s \overset{E}{\sim} s'$ if $se \sim s'e$ holds for all $e \in E$. Based on these relations, S can be partitioned into *compatibility classes* which are groups of

statistically similar samples. Unlike in traditional L*, where each sample translates to a state in the derived automaton, each state of the derived PRM will correspond to a representative from a compatibility class. Each class is defined via compatibility to its representative. Note that the compatibility relation is not an equivalence relation; a given sample may be compatible with several representatives and two elements of a compatibility class may be incompatible. To resolve these ambiguities, define the *rank function* $\mathsf{Rank} : S \to \mathbb{N}$ as

$$\mathsf{Rank}(s) = \sum_{\lambda \in \Lambda} \sum_{\gamma \in \Gamma} T_{s\lambda}(\gamma),$$

to quantify the amount of information about each sample contained in the table. The unique *representative* of s is then defined as the compatible sample of maximal rank and is characterized by the function $\mathsf{Rep} : S \to S$, given as

$$\mathsf{Rep}(s) = \operatorname*{arg\,max}_{s' : s \overset{E}{\sim} s'} \mathsf{Rank}(s').$$

Denote by $\mathsf{Rep}(S)$ the set $\{\mathsf{Rep}(s) : s \in S\}$ of all representatives.

Definition 4.10 – Sampling Closure & Sampling Consistency

A sampling observation table is

CLOSED iff, for all $s\lambda\gamma \in S\Lambda\Gamma$ with $T_{s\lambda}(\gamma) > 0$, there exists $r \in \mathsf{Rep}(S)$ such that $s\lambda\gamma \overset{E}{\sim} r$ holds;

CONSISTENT iff, for all compatible pairs in the set $\{(s, s') \in S \times S : s \overset{E}{\sim} s'\}$ and all $\lambda, \gamma \in \Lambda \times \Gamma$, at least one of $T_{s\lambda}(\gamma) = 0$ or $T_{s'\lambda}(\gamma) = 0$ holds, or $s\lambda\gamma \overset{E}{\sim} s'\lambda\gamma$.

With the above definitions established, we are now in position to describe the high-level flow of the learning procedure, which is specified explicitly in Algorithm 2. While the details of the sub-procedures are significantly different from those involved in traditional L*, the overall structure of the process is quite similar. The algorithm starts with an empty observation table and enters the loop at line 2. Since the table is empty to begin with and there does not exist a counterexample, the conditional at line 3 and the inner loop are bypassed, and an equivalence query is executed at line 20. Deferring the details for now, the equivalence query builds a hypothesis PRM from the observation table and challenges the teacher to find a counterexample showing a discrepancy between the hypothesis and the true reward function. If no counterexample is found, the halting condition at line 21 is met and the algorithm

Algorithm 2: $L^\star$-Based Active Inference for PRM Representations of Stochastic Non-Markovian Reward Signals.

1 Initialize observation table $\langle S, E, T\rangle$ by setting $S := \{\varepsilon\}$ and $E := \{\varepsilon\}$;
2 **repeat**
3 **if** exists counterexample $\chi = \lambda_1\gamma_1 ... \lambda_n\gamma_n$ **then**
4 `prefixes` $:= \{\lambda_1\gamma_1 ... \lambda_k\gamma_k : 1 \leq k \leq n\}$;
5 `Queries` $:=$ `prefixes`$(\Lambda\Gamma \cup \{\varepsilon\})E$;
6 $S := S \cup$ `prefixes`;
7 **repeat**
8 **if** observation table is inconsistent **then**
9 Choose two prefixes $s, s' \in S$, a label $\lambda \in \Lambda$, a reward $\gamma \in \Gamma$, and a suffix $e \in E$ such that $s \overset{E}{\sim} s'$ and $\text{Diff}_T(s\lambda\gamma e, s'\lambda\gamma e)$.;
10 $E := E \cup \{\lambda\gamma e\}$;
11 `Queries` $:=$ `Queries` $\cup\, (S(\Lambda\Gamma \cup \{\varepsilon\})\{\lambda\gamma e\})$;
12 **else if** observation table is open **then**
13 Choose a string $s \in S$, label $\lambda \in \Lambda$, and reward $\gamma \in \Gamma$ such that $T_{s\lambda}(\gamma) > 0$ and $s\lambda\gamma \overset{E}{\nsim} r$ for all $r \in \text{Rep}(S)$;
14 $S := S \cup \{s\lambda\gamma\}$;
15 `Queries` $:=$ `Queries` $\cup\, (\{s\lambda\gamma\}(\Lambda\Gamma \cup \{\varepsilon\})E)$;
16 **foreach** $\zeta \in$ `Queries` **do**
17 $T :=$ `MembershipQuery`(ζ, T);
18 `Queries` $:= \emptyset$;
19 **until** observation table is closed and consistent;
20 $T, \chi :=$ `EquivalenceQuery`$(\mathcal{O})$;
21 **until** *there exists no counterexample* χ;

terminates.

If a counterexample is found, it is returned and stored in the variable χ, and we return to the beginning of the loop. In this case, the conditional block at line 3 is entered and the counterexample χ and all of its prefixes are added to the set of samples S of the observation table. Additionally, we would like to gain more information about the counterexample, so we save it and its extensions in the variable $Queries$.

Next, the loop at line 7 is entered, in which the table is checked for consistency. If the table is inconsistent (lines 8-11), a witness $\lambda\gamma e$ is found, added to the set E of experiments, and the set of all samples extended by this witness is added to the set $Queries$. When the table is consistent, we proceed (lines 12-15) by checking if it is closed. If it is not closed, we similarly find a witness, this time $s\lambda\gamma$, add it to the sample set S and add its extensions by the set of all experiments to the set $Queries$. For each

string in $Queries$ a membership query is executed to gather more data into the table (lines 16-17). The variable $Queries$ is then emptied (line 17), and this sub-process (lines 7-19) is iterated until the observation table is both closed and consistent. At this point, a hypothesis PRM can be constructed again, and the entire process repeats until the equivalence query fails to produce a counterexample to the current hypothesis. Algorithm 2 always learns a reward-deterministic PRM, even if the true reward is specified as a non-reward-deterministic PRM.

Equivalence Queries

In order to construct a hypothesis PRM, we need a statistically significant amount of data about system trajectories to be stored in the observation table. We assume that a threshold parameter $N \in \mathbb{N}$ specifying the minimum number of samples necessary to estimate a transition probability is provided. We use $\perp$ to denote a sink state to which transitions are diverted when the data is insufficient for estimating transition probabilities otherwise.

Definition 4.11 – Hypothesis Reward Machine

Given a closed and consistent observation table $\langle S, E, T\rangle$, we construct a PRM $H = \langle \Lambda, \Gamma, Q, q_I, \delta, \rho\rangle$ as follows:

- $Q = (\mathsf{Rep}(S) \times \Gamma) \cup \{q_I, \perp\}$;
- $q_I = \langle \varepsilon, 0\rangle$;
- $\delta(q,\lambda,q') = \begin{cases} 0 & \text{if } q = \perp \text{ and } q' \neq \perp, \\ 1 & \text{if } q' = \perp \text{ and either } q = \perp \text{ or } \sum_{g\in\Gamma} T_{q\lambda}(g) < N, \\ \dfrac{T_{q\lambda}(\gamma)}{\sum_{g\in\Gamma} T_{q\lambda}(g)} & \text{if } \sum_{g\in\Gamma} T_{q\lambda}(g) \geq N \text{ and } q = \langle r,\eta\rangle \text{ and } q' = \langle \mathsf{Rep}(r\lambda\gamma),\gamma\rangle, \end{cases}$
- $\rho(q,\lambda,q') = \begin{cases} 0 & \text{if } q = \perp \text{ or } q' = \perp, \\ \gamma & \text{if } q' = \langle \mathsf{Rep}(r\lambda\gamma),\gamma\rangle \text{ and } q = \langle r,\eta\rangle. \end{cases}$

Equivalence queries are one of two interfaces between the learner and the teacher. In our setting, the teacher does not have complete knowledge of the system under learning, and so the equivalence query amounts to repeatedly executing the system and the hypothesis in tandem, trying to find inconsistencies between the two. Algorithm 3 specifies the procedure, which takes a closed and consistent

Algorithm 3: Equivalence Query

```
   Data: N_stop
1  Function EquivalenceQuery(𝒪):
2      Make hypothesis H from table 𝒪, initialize 𝒬_H.;
3      flag := false;
4      for 1 ≤ i ≤ N_stop do
5          λ, 𝒬_H, flag := Query(𝒬_H, H, equiv);
6          for prefix λ_1 r_1 ... λ_k r_k of z do
7              z_k := λ_1 r_1 ... λ_k;
8              T_{z_k}(r_k) := T_{z_k}(r_k) + 1;
9          if flag then
10             return T, z;
11     return T, None;
```

observation table as its single parameter. It begins on line 2 by constructing a hypothesis PRM according to Definition 4.11 and initializing the corresponding Q-table $\mathcal{Q}_H$. The variable `flag` is initially set to `false` and will be set to true if the teacher finds a counterexample, *i.e.* a label-reward string impossible under the hypothesis PRM, including those leading to the failure state $\perp$. Additionally, each call to the teacher (line 5) returns a label-reward string z and an updated Q-table $\mathcal{Q}_H$. Regardless of whether z is a counterexample, its data is added to the observation table (lines 6-8). Next (lines 9-11), we check if `flag` is set to `true`, indicating z is a counterexample, and if so, the updated table is returned along with z, ending the loop early. Otherwise, the loop repeats, and halts after N_{stop} iterations without encountering a counterexample. In this case, the updated table is returned and the "counterexample" returned is `None`.

Membership Queries

Membership queries are the other interface between learner and teacher, and are designed to gather data on strings that are witnesses to the observation table's openness or inconsistency. Following [Xu+21], we perform a membership query for each such witness string using a pseudo-"reward machine", defined below.

Algorithm 4: Membership Query

Data: N_{query}

1 **Function** `MembershipQuery(`ζ, T`)`**:**
2 Construct query machine H_ζ and initialize $\mathcal{Q}_\zeta$.;
3 **for** $1 \leq i \leq N_{query}$ **do**
4 $z, \mathcal{Q}_\zeta :=$ `Query(`$\mathcal{Q}_\zeta, H_\zeta$`,`member`)`;
5 **foreach** prefix $\lambda_1 r_1 ... \lambda_k r_k$ of z **do**
6 $z_k := \lambda_1 r_1 ... \lambda_k$;
7 $T_{z_k}(r_k) := T_{z_k}(r_k) + 1$;
8 **return** T;

Definition 4.12 – Membership Query Machine

Given a query string $\zeta = \lambda_1 \gamma_1 ... \gamma_{n-1} \lambda_n$, the corresponding membership query machine is $H_\zeta = \langle \Lambda, \Gamma, Q, q_I, \delta, \rho \rangle$ where

- $Q = \{q_0, q_1, ..., q_n\}$;
- $q_I = q_0$;
- $\delta(q' \mid q_k, \lambda) = \begin{cases} 1 & \text{if } \lambda = \lambda_{k+1} \text{ and } q' = q_{k+1} \text{ or if } \lambda \neq \lambda_{k+1} \text{ and } q' = q_k, \\ 0 & \text{otherwise,} \end{cases}$
- $\rho(q_k, \lambda, q') = \begin{cases} 1 & \text{if } \lambda = \lambda_{k+1} \text{ and } q' = q_{k+1}, \\ 0 & \text{otherwise.} \end{cases}$

Reward machines incentivize exploration along paths that produce the queried trace. Algorithm 4 gives the membership query specification, which takes as parameters a query string ζ and the map T from the observation table. First (line 2), a membership PRM H_ζ is constructed for ζ, according to Definition 4.12 and a corresponding Q-table $\mathcal{Q}_\zeta$ is initialized . Next, the learner passes the membership PRM and the Q-table to the teacher (line 4) and is given back a label-reward string λ and the updated Q-table. As in equivalence queries, the data contained in λ is added to the observation table (lines 5-7), and the process repeats N_{query} times, where N_{query} is a contextual parameter, known a priori. Finally, the modified observation table is returned (line 8).

Teaching By Reinforcement Learning

The function QUERY used in both equivalence and membership queries belongs to the teacher and encapsulates the RL portion of the algorithm. It leverages the fact that the product of a TMDP and a PRM results in a standard MDP.

Any convergent RL algorithm working on the product of the TMDP and membership reward machine, will converge to a policy generating the desired membership query with optimal probability. The RL algorithm optimizes over the product of the TMDP and the membership reward machine, and continues to observe the reward signals from the PRM. In doing so, the policy that maximizes these rewards is guided towards answering the underlying membership query by finding a trajectory of states in the decision process whose induced trace of labels answers the membership query. Similarly, any convergent RL algorithm working on the product of the TMDP and hypothesis reward machine, will converge to an optimal policy with respect to the rewards consistent with the hypothesis reward machine. If the hypothesis correctly encodes the true reward function, we get an optimal policy for the environment.

The RL-driven query is given in Algorithm 5, which takes a Q-table, a reward machine, and `type` indicating whether the query is for membership or equivalence. Furthermore, the procedure assumes access to contextual parameters ϵ for choosing the next action (line 4), α and β as the learning rate and discount factor, respectively, used in the Q-table updates (lines 9 and 11), and N_{ep} as the episode length of each RL execution. Note that the teacher interacts with the environmental TMDP and the given PRM individually, but by doing so in tandem, it simulates the execution of their product. It treats both M and H as black-boxes in the sense that it may only observe the current state and outputs of each after executing them with particular inputs using the STEP function. It is also assumed that the environment M resets to its initial state after the QUERY function returns.

Now we have a complete picture of the RL* algorithm. A flow chart outlining the learning process is provided in Figure 4.2[1] .

Convergence

We conclude this chapter by establishing convergence for the RL* algorithm. A crucial lemma for our argument is the following.

[1] Diamonds represent conditional branching points. Green outgoing edges from diamonds indicate the flow if the condition evaluates to true, while red outgoing edges indicate the flow when the condition evaluates to false.

Algorithm 5: Generic RL-driven Query

Data:
- episode length N_{ep},
- tolerance ϵ,
- discount factor β,
- learning rate α

Function `Query`$(\mathcal{Q}, H, \texttt{type})$**:**
  $x, q, z, \texttt{flag} := x_I, q_I, \varepsilon, \texttt{false}$;
  for $1 \leq i \leq N_{ep}$ **do**
    $a := \texttt{EpsilonGreedyAction}(\mathcal{Q}, x, q, \epsilon)$;
    $x', r := \texttt{Step}(M, x, a)$;
    $z := z\ell(x, a, x')r$;
    $q', \gamma := \texttt{Step}(H, q, \ell(x, a, x'))$;
    if type = member **then**
      $\mathcal{Q}(q, x, a) := (1-\alpha)\mathcal{Q}(q, x, a) + \alpha\big(\gamma + \beta \max_{a' \in A} \mathcal{Q}(q', x', a')\big)$;
    else if type = equiv **then**
      $\mathcal{Q}(q, x, a) := (1-\alpha)\mathcal{Q}(q, x, a) + \alpha\big(r + \beta \max_{a' \in A} \mathcal{Q}(q', x', a')\big)$;
      if z is a counterexample **then**
        flag := true;
    $x, q := x', q'$;
  if type = member **then**
    return $z, \mathcal{Q}$;
  else if type = equiv **then**
    return $z, \mathcal{Q}$, flag;

Lemma 4.1 – Tappler, Aichernig, Bacci, Eichlseder, and Larsen [Tap+19; Tap+21]

Consider MDPs M and M', both with at most k states, that have the same language $L(M) = L(M')$. If $[\![M]\!](w, l) = [\![M']\!](w, l)$ holds for all $\langle w, l \rangle \in L^{\leq k^2+1}(M)$, then M and M' are equivalent.

The next result follows from the convergence of L* for label-deterministic MDPs, shown in [Tap+19; Tap+21]. In that work, the authors show that under uniformly randomized testing strategies, the sampling-based L* algorithm converges almost surely in the limit to the MDP under learning. Our learning algorithm can be seen as a variant of sampling-based L* in which sampling is implemented via reinforcement learning. By resetting the learning rate parameter to very low values at the start of each query and gradually increasing it after a sufficiently long initial period of exploration results in period of essentially random exploration and thus simulates a uniformly random testing strategy. This allows us to apply the convergence arguments from [Tap+19; Tap+21] to our setting to assert that our learning procedure converges almost surely in the limit to the PRM (or an equivalent PRM) encoding

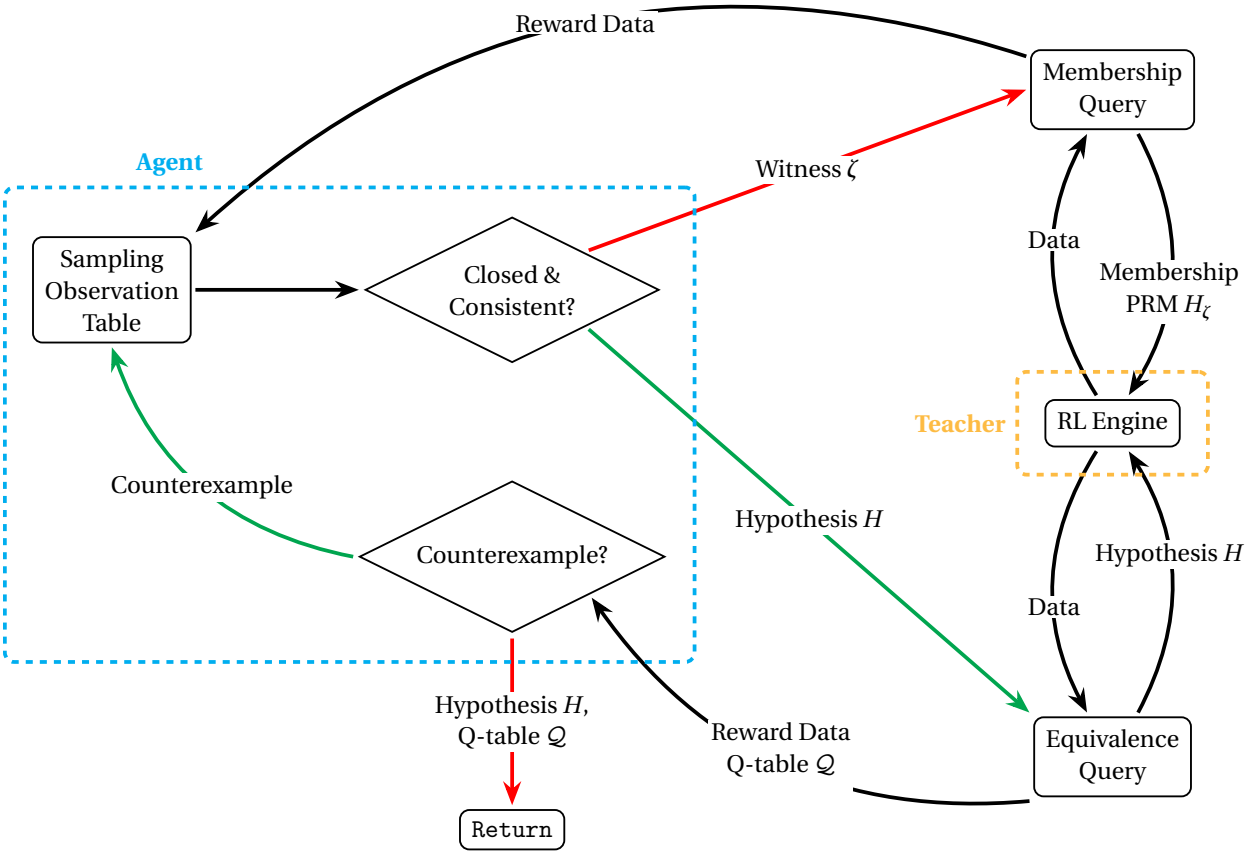


Figure 4.2: A visual schematic of Algorithm 2.

the reward function of the TMDP under learning. In particular, Lemma 4.1 provides a lower bound on the reinforcement learning episode lengths necessary to achieve convergence in the limit.

Theorem 4.3 – Convergence of RL*

Let T be a TMDP with m states, and suppose there exists a reward-deterministic PRM H with k states encoding the reward function of T. Let H_n be the hypothesis PRM passed to the n^{th} equivalence query, and suppose that the episode length is at least $(mk)^2 + 1$.

(1) If every $l \in \Lambda^*$ occurs in a pair $\langle w, l \rangle \in L(T)$, then $H_n \xrightarrow{\text{a.s.}} H$ as $n \to \infty$.

(2) Otherwise, $T \otimes H_n \xrightarrow{\text{a.s.}} T \otimes H$ as $n \to \infty$.

Theorem 4.3 tells us that Algorithm 2 almost surely converges to a PRM H encoding the reward function of the TMDP T under learning, under the assumption that every possible label sequence is attainable from T. If this assumption does not hold, then we cannot claim that the true PRM encoding the

reward function has been learned, since we cannot sample the behavior of the system on unobservable traces. On the other hand, we can say that the learned PRM is equivalent to H, *modulo the language of* T. Given sufficient interactions, the algorithm will almost surely converge to a PRM that is equivalent to H on those label sequences in the language of T. Therefore, optimal strategies for $T \otimes H$ can be learned from the product of T and the learned PRM.

While the primary contributions of this chapter are theoretical, we implemented RL* in Python and ran it on a number of small benchmarks to validate our approach. Recall the office grid-world scenario from Example 4.1. Notice that the PRM from Figure 4.1b is not reward-deterministic, yet RL* learned an equivalent reward-deterministic PRM. The PRM returned by the algorithm is displayed in Figure 4.3.

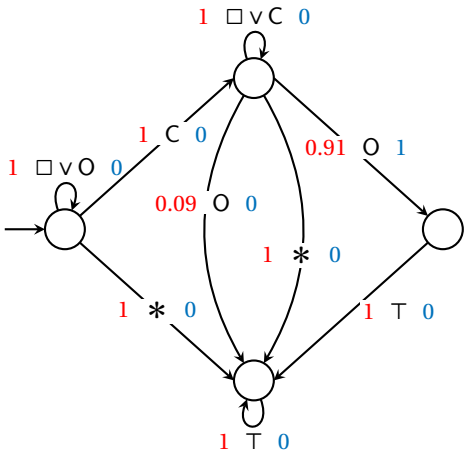


Figure 4.3: A reward-deterministic PRM learned by RL*.

Chapter 5

Regular Markov Decision Processes & Regular Reinforcement Learning

This chapter presents an approach for modeling optimization environments using regular languages and rational transductions. We call MDPs represented this way *regular Markov decision processes* (RMDPs), and we call reinforcement learning in RMDPs *regular reinforcement learning* (RRL).

After defining the model, we establish the following theoretical results.

- The optimal expected payoff, known as the value, of a given RMDP under an arbitrary payoff function is not computable.
- For a given RMDP with computable rewards and transition probabilities, the value under a discounted payoff is approximately computable to any desired tolerance.
- For a given RMDP with computable rewards and transition probabilities, the value under a discounted payoff is PAC-learnable.

Additionally, we identify several sufficient conditions under which an RMPD is provably finite and outline an appropriate adaptation of the Q-learning algorithm for model-free tabular RL for such situations.

Next, we turn our attention towards regular reinforcement learning. We propose a formulation of *deep* RRL that comprises a combination approach which simultaneously utilizes the principles of both deep RL and symbolic RL. The inherent regularity incorporated into RRL informs and directs our approach to deep RL. By viewing regular languages as finite automata and viewing finite automata as labeled directed graphs, we are able to harness the power of graph neural networks (GNNs) for approximating optimal policies when states of the environment are regular languages. Graph neural

networks [Wu+21] are neural network architectures that process graphs as input, typically by performing repeated message passing of vectors over the graph's structure. We demonstrate through a collection of experimental case studies that deep regular RL is both natural and effective.

Regular Model Checking

Our modeling approach for RMDPs is inspired by *regular model checking* (RMC), a framework for formal verification of infinite-state systems, built on the basis of language-theoretic modeling techniques [Abd+04; Abd+12; Bou+00; WB98; Kes+01]. System states are encoded symbolically as regular languages and state-transition relations as rational transductions [Sch76; Eil74a; Ber79]. This modeling approach enjoys an unusual combination of expressivity and structure. On the one hand, it is powerful enough to be computationally universal in the sense that it is possible to simulate arbitrary computation. On the other hand, the built-in regularity provides considerable structure that facilitates the design of automated analysis methods. As evidenced by the RMC literature, a number of approximation schemes and heuristics [Bou+00; JN00; Tou01; DLS01; DLS02; Abd+02; BLW03; BHV04; HV04] have made it a practical verification approach. It has, for example, been applied to verify programs with unbounded data structures such as lists and stacks [Bou+00; Abd+04]. Moreover, since infinite strings over a finite alphabet can be naturally interpreted as real numbers in the unit interval, regular model checking over infinite strings provides a framework [BLW04a; BW02; Leg12; BJW05; BLW04b; LW10] to analyze properties of dynamical systems.

Fundamentally, all of the generic techniques, *i.e.* those not tailored to some specific domain, devised for verification by RMC are possible because of the language-theoretic regularity that is directly incorporated into the modeling framework. A natural question arises: *What can the modeling framework of RMC and its built-in regularity offer to related areas?* We investigate this question in the area of reinforcement learning. As in RMC, the primary application of the language-theoretic modeling framework in our setting is the symbolic representation of states and transitions in the underlying system.

Transductions

Let Σ and Γ be finite alphabets. A relation $\theta : \Sigma^* \times \Gamma^*$, is called a *transduction* [Ber79; Eil74a] . For words $w \in \Sigma^*$ and $w' \in \Gamma^*$, define the image of θ on w as $\theta(w) = \{w' \in \Gamma^* : \langle w, w' \rangle \in \theta\}$ and the inverse image of θ from w' as $\theta^{-1}(w') = \{w \in \Sigma^* : \langle w, w' \rangle \in \theta\}$. The domain of θ is the set $\mathrm{dom}\,\theta = \{w \in \Sigma^* : \theta(w) \neq \emptyset\}$. The image of θ is the set $\mathrm{im}\,\theta = \theta(\mathrm{dom}\,\theta) = \bigcup_{w \in \Sigma^*} \theta(w)$. The *composition* of transductions $\theta_1 : \Sigma_1^* \times \Sigma_2^*$ and $\theta_2 : \Sigma_2^* \times \Sigma_3^*$ is defined as a composition of relations

$$\theta_2 \circ \theta_1 = \{\langle w, w_2 \rangle \in \Sigma_1^* \times \Sigma_3^* : \exists w_1 \in \Sigma_2^* . \langle w, w_1 \rangle \in \theta_1 \text{ and } \langle w_1, w_2 \rangle \in \theta_2\}.$$

Given a finite set Θ of transductions of type $\Sigma^* \times \Sigma^*$, each finite word $\theta_1 \dots \theta_n \in \Theta^*$ corresponds to the rational transduction $\theta_n \circ \dots \circ \theta_1$ where ε represents the identity mapping (i.e. $\varepsilon(x) = x$ for every $x \in \Sigma^*$). We identify the word $\theta_1 \dots \theta_n$ with the transduction $\theta_n \circ \dots \circ \theta_1$ so that $\theta_1 \dots \theta_n(x) = \theta_n \circ \dots \circ \theta_1(x)$ holds for every $x \in \Sigma^*$. The set of languages reachable from a given language L via elements of Θ^* is called the *orbit* of Θ on L and is written as $\Theta^*(L) = \{\tau(L) : \tau \in \Theta^*\}$.

Definition 5.1 – Nondeterministic Finite-State Transducer (NFT)

Syntax: A nondeterministic finite-state transducer T is given by a tuple $\langle \Sigma, \Gamma, Q, q_0, F, \delta \rangle$, where

- Σ is a finite input alphabet;
- Γ is a finite output alphabet;
- Q is a finite set of states;
- $q_0 \in Q$ is a distinguished initial state;
- $F \subseteq Q$ is a set of final or accepting states;
- $\delta : Q \times \Sigma \to 2^{Q \times \Gamma^*}$ is a transition function.

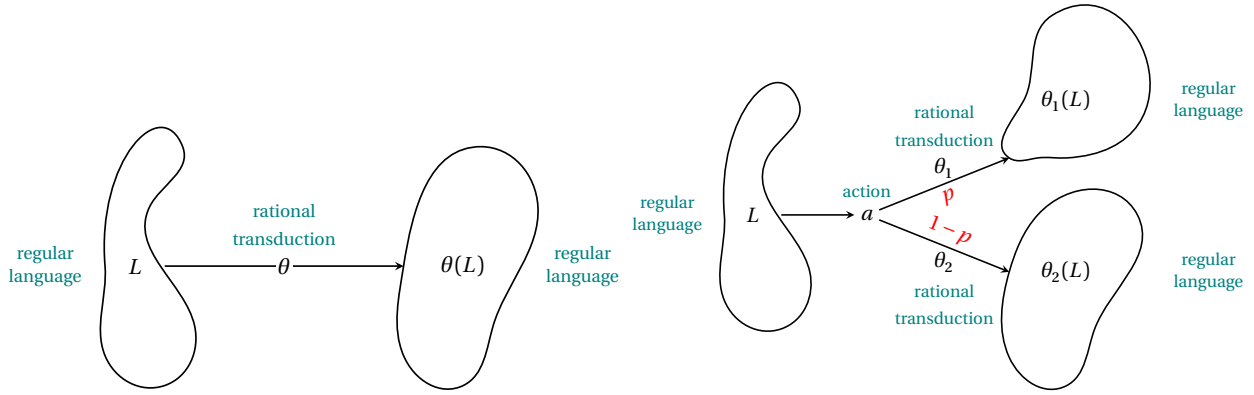
Semantics: Define $\delta^* : Q \times \Sigma^* \to 2^{Q \times \Gamma^*}$ such that $\delta(q, \varepsilon) = \{\langle q, \varepsilon \rangle\}$ and

$$\delta^*(q, \sigma w) = \{\langle q_2, w_1 w_2 \rangle : \langle q_1, w_1 \rangle \in \delta(q, \sigma) \text{ and } \langle q_2, w_2 \rangle \in \delta^*(q_1, w)\},$$

for any state $q \in Q$, symbol $\sigma \in \Sigma$, and word $w \in \Sigma^*$ The transduction $[\![T]\!] : \Sigma^* \times \Gamma^*$ implemented by T is the relation

$$[\![T]\!] = \{\langle w, w' \rangle : \langle q, w' \rangle \in \delta^*(q_0, w') \text{ and } q \in F\}.$$

An NFT T is *functional* (fNFT) if the transduction it implements is a function, *i.e.* $\left|[\![T]\!](w)\right| = 1$ for all $w \in \Sigma^*$. In this case, we write the type of the transduction as $[\![T]\!] : \Sigma^* \to \Gamma^*$. A *deterministic finite-state transducer* (DFT) is an NFT T with a transition function such that $|\delta(q, \sigma)| = 1$ holds for every state



(a) A transition in regular model-checking.

(b) A transition in a regular Markov decision process.

Figure 5.1: An illustration of the difference between system transitions in RMC as compared to RMDPs.

$q \in Q$ and symbol $\sigma \in \Sigma$. For DFTs, we use functional types such as $[\![T]\!] : \Sigma^* \to \Gamma^*$ and $\delta : Q \times \Sigma \to Q \times \Gamma^*$.

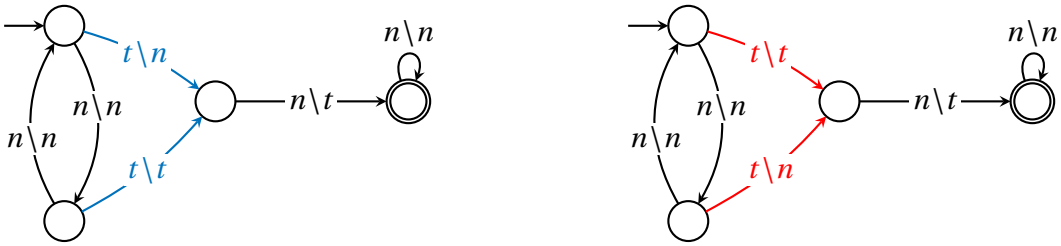
A *rational transduction*—also called a *rational relation*—θ is a transduction for which there exists an NFT T such that $\theta = [\![T]\!]$. A *rational function* is a functional rational relation.

5.1 Regular Markov Decision Processes

Regular Markov decision processes are MDPs[1] where states have been provided with a specific structure expressed through a regular language over some alphabet Σ. An execution of an RMDP starts with an initial regular language $L_0 = I$. At each step $i \geq 0$, a decision maker or learning agent selects an action a_i from the current state L_i. The environment resolves the action by selecting a transduction θ_i from the probabilistic distribution over Θ corresponding to the action and returning the next state as $L_{i+1} = \theta(L_i)$ and returning the reward $r_{i+1} = \mathcal{R}(L_i)$. The goal of the learning agent is to select actions in a manner that optimizes a given objective F.

Before giving a formal definition, we illustrate the concepts involved in RMDPs through the following example, which presents a variation on the canonical token passing protocol from the RMC literature [Bou+00; Abd+04].

[1] In this chapter, we assume that the tuple specifying each MDP includes a distinguished initial state x_0.



(a) T_a (b) T_b

Figure 5.2: Transducers corresponding to actions a and b of Example 5.1.

Example 5.1 – Token Passing

The token passing protocol involves an arbitrary number of processes arranged in a linear topology and indexed by consecutive natural numbers. At any point in time, each process can be in one of two states: t if it has a token or n if it does not. The states of the system are then sets of strings over the alphabet $\{t, n\}$. The initial state, in which only the leftmost process has a token, is the regular language tn^*.
At each time step, an agent chooses an action from the set $\{a, b, c\}$, whose elements to the following outcomes.

(a) Each even-indexed process with a token passes it right. Each odd-indexed process with a token keeps it and makes a copy which it passes right.

(b) Each odd-indexed process with a token passes it to the right. Each even-indexed process with a token keeps it and makes a copy which it passes right.

(c) The outcome of a (resp. b) occurs with probability p (resp. $(1-p)$).

Figure 5.2[a] depicts finite state transducers corresponding to actions a and b.
In the context of RMC, the property of interest for verification is that exactly one process possesses a token at any given time. The essence of this property may be captured by a reward function $R : 2^{\{t,n\}^*} \to \mathbb{R}$ defined such that

$$R(L) = \begin{cases} 1 & \text{if } L \subseteq n^* t n^*, \\ -1 & \text{otherwise.} \end{cases}$$

From the initial configuration tn^*, action a moves the system to state ntn^* and incurs a reward of 0, while action b transitions the system to the configuration ttn^* and incurs a reward of -1. The optimal policy selects action a when the token is with a process with an even index and chooses action b otherwise.

[a] Note that these transducers are designed under the implicit assumption that their input strings will contain at most one t symbol. This is because there is no way of removing tokens from the system once they have been introduced, and, as a result, no learning can occur after a second token is introduced.

Definition 5.2 – Regular Markov Decision Process (RMDP)

Syntax: A regular Markov decision process M is given by a tuple $\langle \Sigma, I, \Theta, A, \mathcal{P}, \mathcal{R} \rangle$, where

- Σ is a finite alphabet;
- $I \subseteq \Sigma^*$ is an initial regular language;
- Θ is a finite set of rational transductions of type $\Sigma^* \times \Sigma^*$;
- A is a finite set of actions;
- $\mathcal{P} : 2^{\Sigma^*} \times A \to \mathrm{Dist}(\Theta)$ is a probabilistic transition function mapping language-action pairs to distributions over Θ;
- $\mathcal{R} : 2^{\Sigma^*} \to \mathbb{R}$ is a bounded reward function.

Semantics: M is interpreted as a countable-state MDP $[\![M]\!] = \langle X, x_0, A, P, R \rangle$, such that

- $X = \Theta^*(I)$;
- $x_0 = I$;
- $P(\theta(L) \mid L, a) = \mathcal{P}(\theta \mid L, a)$, for every language $L \in X$, action $a \in A$, and transduction $\theta \in \Theta$;
- $R(L, a) = \mathcal{R}(L)$, for every language $L \in X$ and every action $a \in A$.

The value of an objective F over an RMDP M is defined by the equation

$$\mathrm{Val}_M(\mathsf{F}, I) = \mathrm{Val}_{[\![M]\!]}(\mathsf{F}, x_0).$$

An RMDP is said to be *computable* if the transition probability map $\mathcal{P}$ and the reward function $\mathcal{R}$ are computable. If the functions can be computed in polynomial time, then the RMDP is *efficiently computable*.

Our first theoretical result establishes that value problems for RMDPs are generally undecidable. Using the terminology of Rice [Ric53], a property is *trivial* with respect to class of models if it holds for all models in the class or if it holds for no models in the class.

Theorem 5.1 – Undecidability

Determining if an arbitrary RMDP satisfies any fixed non-trivial property is undecidable.

Proof. We construct, as depicted in Figure 5.3, a deterministic FST that can simulate the transition relation of an arbitrary Turing machine (TM). Configurations of the TM, *i.e.* combinations of internal state and tape contents, are encoded as words in the regular language $(0+1)(0+1)^* Z (0+1)^*$, where Z is the finite set of internal states. The index i of the single element of Z occurring in each such word represents that the tape head of the TM is at position $i-1$. Assume that the TM in question includes an

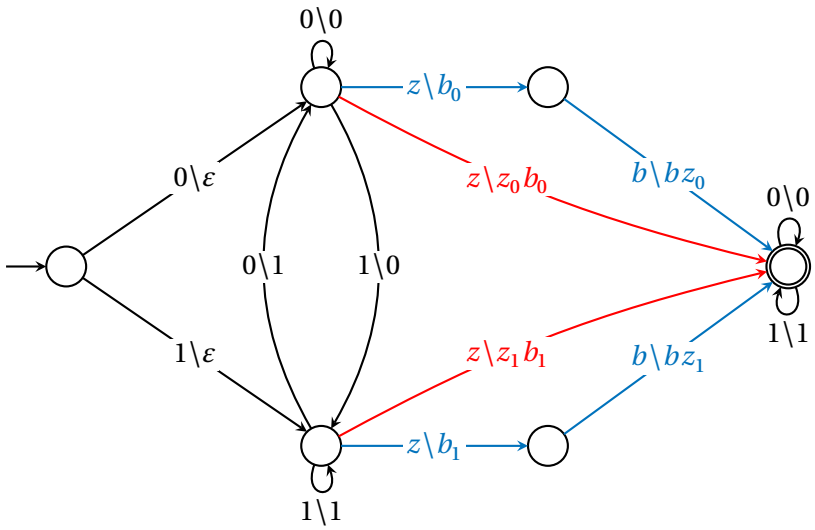


Figure 5.3: The FST from the proof of Theorem 5.1, simulating the transition function of a Turing machine over a binary alphabet.

arbitrary transition instruction according to the following pair of rules.

- If 0 is read in state z, then write b_0, go to state z_0, and move the tape head.
- If 1 is read in state z, then write b_1, go to state z_1, and move the tape head.

We leave the direction of the tape head shift undetermined and show all possibilities in Figure 5.3. The red edges show the construction for the above TM transition when the tape head shifts left. The blue edges show the construction when the tape head shifts right.

In combination, this construction and Rice's Theorem [Ric53]—which states that no non-trivial property is decidable for the class of Turing machines—imply the desired result. □

It follows that optimal values of RMDPs are not computable in general.

Corollary 5.1 – Uncomputability of Optimal Values

For arbitrary objectives, the RMDP value problem is undecidable.

Finitary Regular Markov Decision Processes

The next few propositions provide three sufficient conditions that guarantee finiteness of an RMDP. Fix an arbitrary RMDP $M = \langle \Sigma, I, \Theta, A, \mathcal{P}, \mathcal{R} \rangle$ with semantics $[\![M]\!] = \langle X, x_0, A, P, R \rangle$. The RMDP M

is called *finite* if the orbit $\Theta^*(I)$ is finite.

A transduction $\theta : \Sigma^* \times \Gamma^*$ is (i) *length-preserving* if $|\theta(w)| = |w|$; (ii) *decreasing* if $|\theta(w)| < |w|$; (iii) *non-increasing* if $|\theta(w)| \leq |w|$; (iv) *non-decreasing* if $|\theta(w)| \geq |w|$; (v) *increasing* if $|\theta(w)| > |w|$, for all $w \in \Sigma^*$.

Proposition 5.1

If I is finite, Θ is non-increasing, $|\Sigma| = n$, and $\max_{w \in I} |w| = m$, then $\Theta^*(I) = 2^{O(n^m)}$.

Proof. The statement can be derived from the observation that the longest string possibly appearing in the image $\theta(I)$ of a finite language I under a non-increasing transformation θ is of length $m = \max_{w \in I} |w|$. There are n^m strings of length m over an alphabet Σ of size n, so it follows that

$$|\theta(I)| \leq 1 + \sum_{k=1}^{m} n^k.$$

More succinctly, this says that $|\theta(I)| = O(n^m)$. Since the orbit $\Theta^*(I)$ must comprise some collection of subsets of this image, we conclude that $|\Theta^*(I)| = 2^{O(n^m)}$. □

A transduction $\theta : \Sigma^* \times \Sigma^*$ is (i) *specializing* if $\theta(L) \subseteq L$; (ii) *non-specializing* if $\theta(L) \not\subseteq L$; (iii) *generalizing* if $L \subseteq \theta(L)$; (iv) *non-generalizing* if $L \not\subseteq \theta(L)$, for all $L \subseteq \Sigma^*$.

Proposition 5.2

If $|I| = n$ and Θ is specializing, then $\Theta^*(I) \leq 2^n$.

Proof. This result can be deduced from the observation that when beginning with a finite initial language, specializing transformations can only generate languages with a cardinality that is either equal to or smaller than that of I. □

Let $\sim_M \subseteq 2^{\Sigma^*} \times 2^{\Sigma^*}$ be an equivalence relation over languages such that $L_1 \sim_M L_2$ iff

$$\mathcal{R}(L_1) = \mathcal{R}(L_2) \quad \text{and} \quad \forall \theta \in \Theta. \quad \theta(L_1) \sim_M \theta(L_2).$$

This relation is often useful as a means of partitioning the state space of an RMDP into a finite set of equivalence classes that respects the structure of its dynamics. For instance, it is straightforward to deduce the following proposition.

Proposition 5.3

If there exists $n \in \mathbb{N}$ such that $\mathcal{R}(L) = \mathcal{R}\big(L \cap \Sigma^{\leq n}\big)$ holds for every $L \subseteq \Sigma^*$, then $\Theta^*(I)$ has finitely many $\sim_M$-equivalence classes.

5.2 Discounted Optimization In Regular Markov Decision Processes

Let $\mathfrak{r} = (r_n)_{n\in\mathbb{N}}$ be a bounded infinite sequence of real numbers. Recall that, for a discount factor $\lambda \in [0,1)$, the *λ-discounted objective* is defined as

$$\mathsf{D}_\lambda(\mathfrak{r}) = \sum_{n=1}^{\infty} r_n \lambda^{n-1}.$$

Over computable RMDPs, it is possible to approximate the D_λ-value to an arbitrary tolerance ϵ. This result is facilitated by properties of the discounted objective and therefore holds even when the RMDP in question is not finite.

Theorem 5.2 – Approximability of Discounted Values

If M is a computable RMDP, then the D_λ-value is ϵ-approximable, for any $\lambda \in [0,1)$ and any $\epsilon > 0$.

Proof. The proof makes use of the standard technique of finding, given ϵ and λ, the least n such that $\left|\mathsf{D}_\lambda(r) - \sum_{k=1}^{n} \lambda^{k-1} r_k\right| \leq \epsilon$. For a given RMDP M, the D_λ-value can be characterized by the following Bellman optimality equations:

$$D(L) = \max_{a\in A} \mathcal{R}(L) + \lambda \sum_{\theta\in\Theta} \mathcal{P}(\theta \mid L, a) D(\theta(L)).$$

It follows from the more general result [Fei11] for MDPs with countable state spaces, finite action spaces, and bounded rewards. Let B be an upper bound on the absolute value of the rewards. For a given $\epsilon > 0$, let n be such that $\dfrac{\lambda^{n+1} B}{(1-\lambda)} \leq \epsilon$. Then a solution to the following recurrence characterizes an ε-optimal value and corresponding memoryful policy for the RMDP:

$$D_n(L) = \begin{cases} \max_{a\in A} \mathcal{R}(L) + \lambda \sum_{\theta\in\Theta} \mathcal{P}(\theta \mid L, a) D_{n-1}(\theta(L)) & \text{if } n > 0 \\ 0 & \text{otherwise.} \end{cases}$$

□

An RL algorithm is *probably approximately correct* (PAC) [Val84], with respect to parameters $\epsilon > 0$ and $\delta > 0$, if after polynomially many samples of the environment, it produces an ϵ-optimal policy with probability $1 - \delta$. Objective functions under which PAC algorithms exist are called PAC-learnable.

Theorem 5.3 – PAC-Learnability of Discounted Values

For every computable RMDP, discounted values are PAC-learnable.

Proof. Our approach for calculating ϵ-optimal policies for the discounted objective involves computing a policy that is optimal for a fixed number of steps, denoted by n. Given $\epsilon > 0$, we choose n such that $\frac{\lambda^{n+1}B}{(1-\lambda)} \leq \epsilon$. This policy can be computed on a finite-state MDP obtained by unfolding the RMDP n times. We can then apply existing PAC algorithms to compute an $\frac{\epsilon}{2}$-optimal policy, which is also an ϵ-optimal policy for the RMDP. □

An *efficiently* PAC algorithm is a PAC algorithm that also runs in polynomial time.

Corollary 5.2

For any efficiently computable RMDP, discounted values are efficiently PAC-learnable

5.3 Regular Reinforcement Learning

We have discussed some conditions that ensure the finiteness of $[\![M]\!]$. When any such condition is satisfied, it becomes feasible to employ off-the-shelf RL algorithms to approximate the D_λ-value. Equation (5.1)—in which $[L]_\sim$ denotes the equivalence class to which the language L belongs—provides an iteration scheme for a variation on the Q-learning [WD92] algorithm tailored for RMDPs. If learning rates $(\alpha_n)_{n\in\mathbb{N}}$ are such that $\sum_{n=1}^{\infty} \alpha_n = \infty$ and $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$, and the trajectory $\big([L_n]_\sim, a_n, r_n\big)_{n\in\mathbb{N}}$ includes each pair $[L]_\sim, a$ infinitely often, then the Q-table obtained by iterating Equation (5.1) converges almost surely to an optimal policy.

$$\mathcal{Q}_{n+1}([L_n]_\sim, a_n) := (1-\alpha_n)\mathcal{Q}_n\big([L_n]_\sim, a_n\big) + \alpha_n\Big(r_n + \max_{a\in A} \mathcal{Q}_n\big([L_{n+1}]_\sim, a\big)\Big) \tag{5.1}$$

Generally speaking, RMDPs may have infinite state spaces, so we cannot guarantee convergence of Q-learning to the optimal value. In light of this fact and the uncomputability of exact discounted values—established by Theorem 5.1—, it makes sense to move towards approximate learning approaches. In the remainder of this section, we develop a deep RL approach, based on *graph neural networks* (GNNs), in order to approximate discounted values for general RMDPs. The key insight enabling deep RL in this context is the observation that we can use automata representations of the states of an RMDP directly as inputs to GNNs. Just as standard deep RL uses feature vectors as inputs for neural networks, our approach uses automata—which are essentially finite labeled graphs—as inputs for GNNs. We term this approach *deep regular reinforcement learning*.

- In *deep RL* a policy π : States $\to$ Actions is approximated by a *neural network*

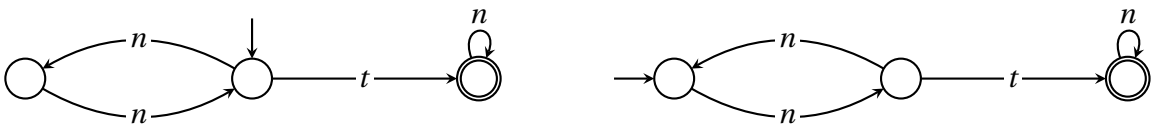
$$N : \text{FeatureVectors} \to \text{Actions}.$$

- In *Deep Regular RL* a policy π : States $\to$ Actions is approximated by a *graph neural network*

$$G : \text{FiniteAutomata} \to \text{Actions}.$$

For our GNN architecture, we select the graph convolution operator proposed by [KW17]. We perform an independent graph convolution for each letter in the DFA, only allowing the convolution to operate over the graph connectivity for that letter, and take the mean of the resulting vectors for each node, followed by a non-linearity. We repeat this for N layers and then concatenate the sum of all node vectors with the element-wise maximum of all node vectors. Then, separate multi-layer perceptrons produce the policy and state value predictions from this representation. We use proximal policy optimization (PPO) [Sch+17] for training. The initial embedding for each node in the DFA is a binary vector of length two, which encodes whether a node is the initial state, and if a node is accepting. For all experiments, the graph neural network had 3 graph convolution layers with hidden dimension 256, and the multi-layer perceptron heads had 2 layers each with hidden dimension 256. We used the LeakyReLU non-linearity.

The remainder of this section presents a number of case studies of regular RL problems and gives experimental results to demonstrate the effectiveness of deep regular RL.



(a) Even DFA (b) Odd DFA

Figure 5.4: DFAs used to represent the equivalence class for even and odd.

Token Passing

We first consider the token passing scenario of Example 5.1. Note that this example admits a partitioning of the environment via the equivalence relation defined in Section 5.1: the case of an even or an odd number of n before the first t forms two equivalence classes. We compare using the GNN on the original representation ("GNN") and on the finite representation formed by the two equivalence classes ("GNN+EC"). Figure 5.4 shows the DFA representations used for the two equivalence classes. The hyperparameters we used for PPO in this case study were 1024 steps per update, 4 optimization epochs, a batch size of 512, a clip range of $\epsilon = 0.2$, and a discount factor of $\lambda = 0.99$. The learning curves are shown in Figure 5.5, in which the dark lines are means and the shaded regions are the 10^{th} to 90^{th} percentiles over 5 training runs.

Note that under the network architecture selected, determining whether the number of n's before the first t is even or odd is largely determined by the multi-layer perceptron components. Roughly, the number of n's before the first t is encoded in unary in the global sum component of the graph representation. The multi-layer perceptrons must then detect parity on this unary representation, which is challenging. To encourage learning, we use a denser reward of 1 on every successful step and -1 on failure, up to a time limit of 30. Although alternative network architectures have the potential to perform better, the simple two-state equivalence class representation is still expected to result in faster learning than the unmodified representation. The learning run is shortened, and the maximum episode length is set to 30 to highlight the difference between these two setups. We see that forming equivalence classes leads to an increase in learning speed, as expected.

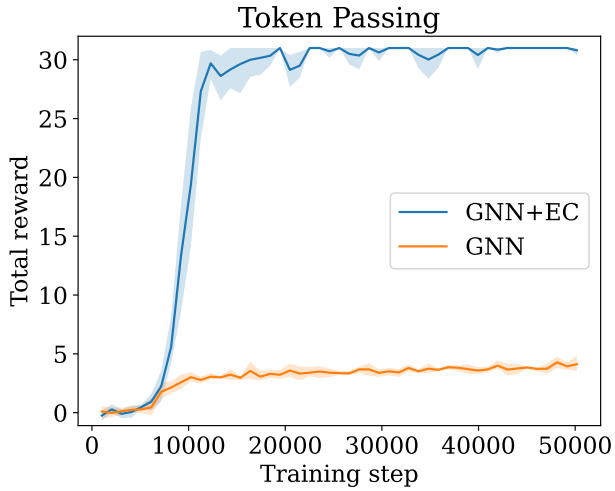


Figure 5.5: Reward curve for the token passing case study.

Duplicating Pebbles

Consider a grid world with multiple pebbles on it. The agent can select two adjacent directions, *e.g.*, "up" and "right", and every pebble will be duplicated and moved in each of these directions. The goal of the agent is to have at least one pebble reach the goal state, while all pebbles do not accumulate a cost greater than a threshold $t = 2$. If a pebble goes over a trap cell, it incurs a cost of 1 for that pebble. If a pebble moves off of the grid, it wraps back around to the other side of the grid.

Although the number of pebbles grows exponentially, doubling after each action, the set of paths the pebbles take has a linear DFA representation. Namely, one can represent the growing paths by adding a state to the DFA with two transitions to that state corresponding to the two directions selected. This added state is marked as the only accepting state. The language of this DFA corresponds to all of the paths that pebbles have taken. The grid layout is shown in Figure 5.6, where the initial pebble begins in the top left, traps are denoted by red cells, and the goal is denoted by a green cell. The number in a cell counts the pebbles it contains. A reward of -1 is given on failure and a reward of 1 is given on success. The hyperparameters we used for PPO in this case study were 512 steps per update, a batch size of 128, 4 optimization epochs, a clip range of $\epsilon = 0.2$, and a discount factor of $\lambda = 0.99$. The resulting reward curve is shown in Figure 5.7, where the dark lines are means and the shaded regions are the 10^{th} to 90^{th}

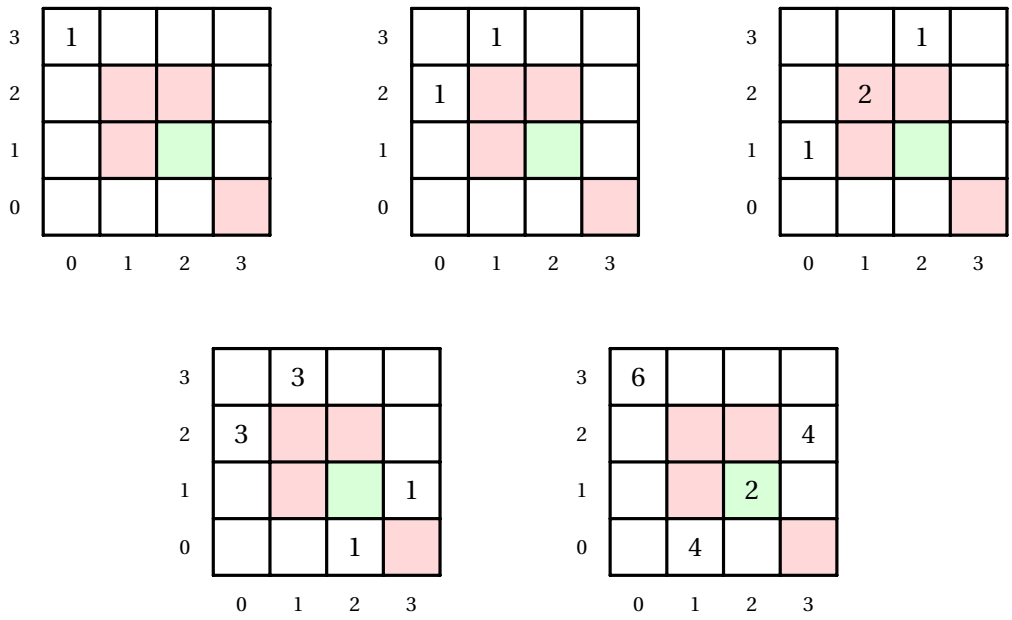


Figure 5.6: Execution of the optimal policy for the duplicating pebbles case study, from left to right and top to bottom.

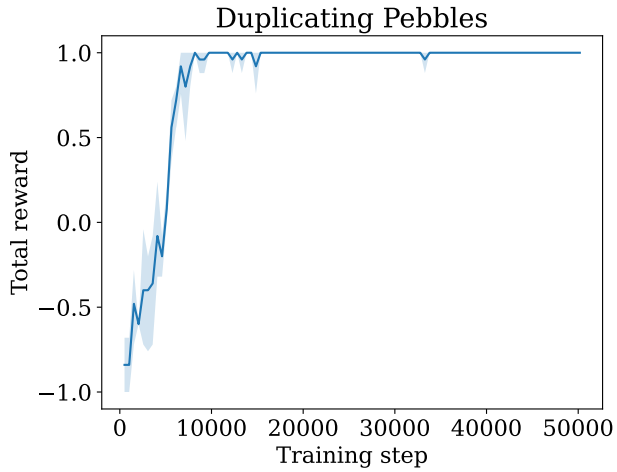


Figure 5.7: Reward curve for the duplicating pebbles case study.

percentiles over 5 training runs. The agent learns the optimal policy “down, right”, “down, right”, “up, left”, “up, left” in about 10k training steps.

Figure 5.6 shows the execution of the optimal policy. Since the representation of the state is a DFA of the possible trajectories, a linear program is solved at each step to find the highest cost path needed for computing the reward. Note that when “up, left” is first performed, some pebbles wrap around to

State	Action
3*3+7+5##	Move
*3+7+5##3	Push
3+7+5#*#3	Move
+7+5#*#33	Pop
+7+5##33*	Push
7+5#+#33*	Move
+5#+#33*7	Pop
+5##33*7+	Push
5#+#33*7+	Move
#+#33*7+5	Pop
##33*7+5+	

State	Action
1+7*8+0##	Move
+7*8+0##1	Push
7*8+0#+#1	Move
*8+0#+#17	Push
8+0#+*#17	Move
+0#+*#178	Pop
+0#+#178*	Pop
+0##178*+	Push
0#+#178*+	Move
#+#178*+0	Pop
##178*+0+	

State	Action
7-1*3##	Move
-1*3##7	Push
1*3#-#7	Move
*3#-#71	Push
3#-*#71	Move
#-*#713	Pop
#-#713*	Pop
##713*-	

Figure 5.8: Examples runs produced by the learned policy for the shunting yard algorithm.

the opposite side of the grid. If "down, right" was performed 3 times, instead of twice, then the agent would fail the objective since the 2 pebbles at $(1,2)$ on the grid would duplicate and visit the trap state again after having already visited it once.

Shunting Yard Algorithm

As described in the introduction, representing the algorithms with stacks and queues is straightforward in regular RL. We consider learning the shunting yard algorithm [Dij61] which transforms an expression in infix notation to postfix notation. We represent the input as a regular language consisting of a single string containing the concatenation of the infix notation input, the stack, and the output, each separated by a special symbol "#". The agent has three actions for moving the first character of the input to the output, pushing the first character of the input to the stack, and popping the top character on the stack to the output. We generate random infix notation expressions and give a reward of -1 if the output is an invalid postfix expression, a reward of 0.5 if the output is a valid postfix expression that evaluates to the wrong value, and a reward 1 if the output evaluates to the correct value. The agent is able to learn an effective strategy in about 100k time steps.

Figure 5.8 shows example runs produced by the learned policy. The representation used during learning is a DFA accepting a single string, but we print the string for clarity. Actions are the actions

selected upon observing that state. The last state is the final state at termination.

The hyperparameters we used for PPO in this case study were 1024 steps per update, 10 optimization epochs, a batch size of 128, a clip range of $\epsilon = 0.2$, and a discount factor of $\lambda = 0.99$. The resulting reward curve is shown in Figure 5.9, in which the dark lines are means and the shaded regions are the 10^{th} to 90^{th} percentiles over 5 training runs.

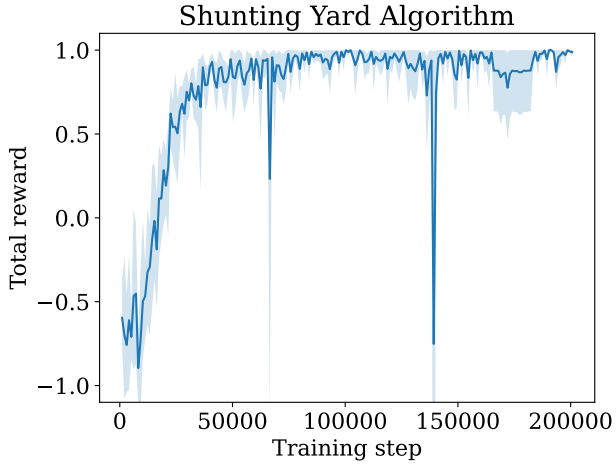
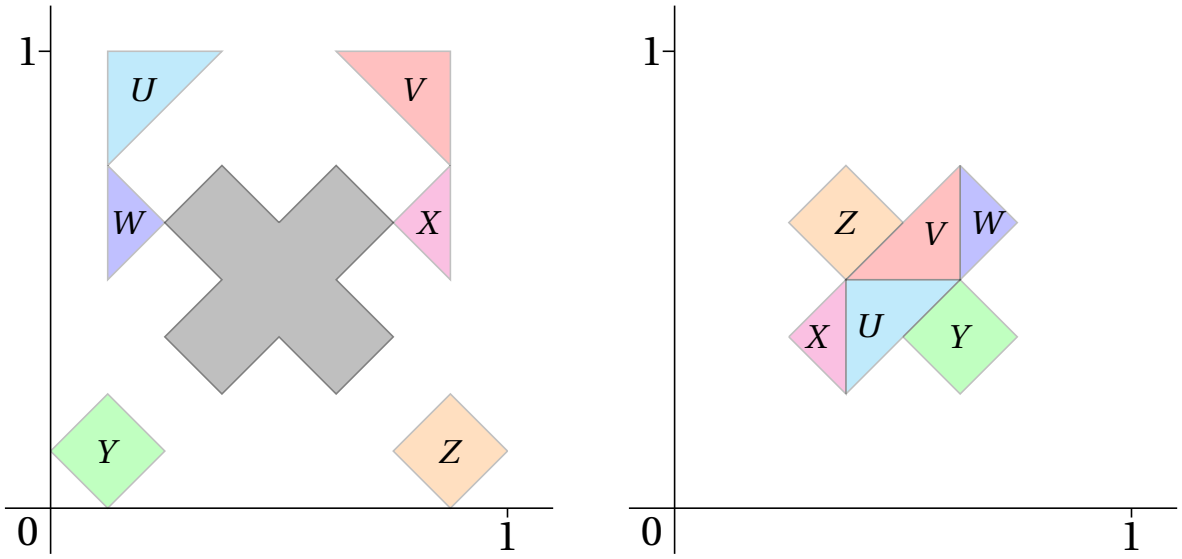


Figure 5.9: Reward curve for the shunting yard case study.

Modified Tangrams

This case study examines the application of deep RRL to variations of geometric puzzles known as tangrams. A tangram is a popular type of puzzle that involves arranging a finite set of polygonal wooden tiles on a flat surface to create a given picture. The picture is typically a silhouette in the shape of a common object such as a building or a tree, and the puzzle is completed once the tiles have been arranged into a configuration that covers the silhouette exactly. A standard tangram set includes a collection of target pictures, 5 right triangles (2 large, 1 medium, and 2 small), a square, and a parallelogram. We consider modified tangrams which we qualify as such because the pieces do not coincide with the standard tile set. In particular, we use the modified tangram depicted in Figure 5.10, where the objective is to completely cover the gray shape at the center resembling the symbol "×" by rearranging the colored tiles $\{U, V, W, X, Y, Z\}$.



(a) Tiles in their initial positions.

(b) Tiles arranged in a solution.

Figure 5.10: A modified tangram.

In order to cast these sorts of puzzles into the RRL framework, we apply standard notions used in positional numeration systems to connect geometric shapes and regular languages. More precisely, tiles are considered as sets of points in the unit square of the Euclidean plane, which can be encoded by digital expansions. Then, sets of points are modeled by regular languages consisting of digital expansions.

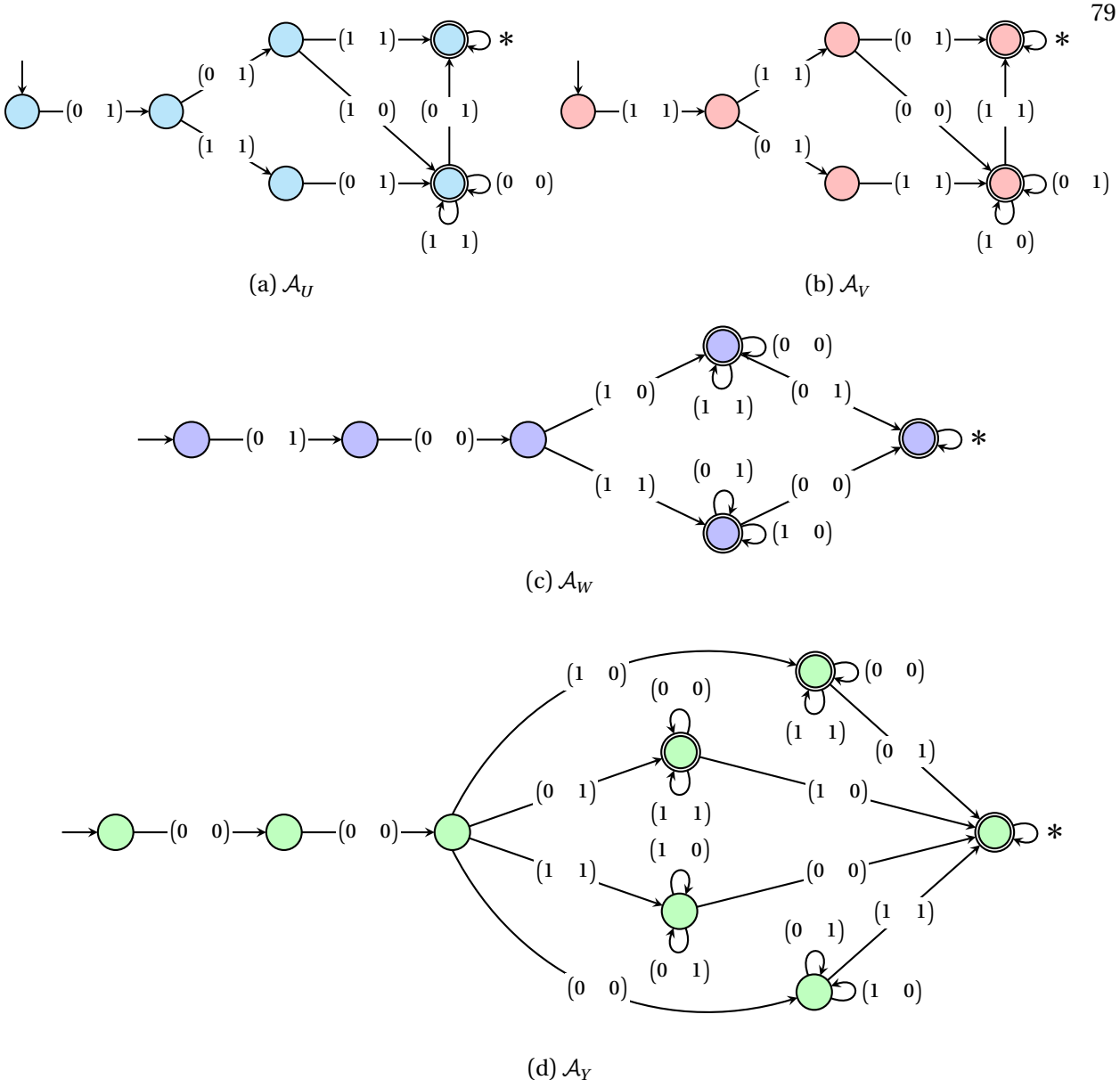
For a numeration base $b \in \mathbb{N}$, define a function $\langle\langle\cdot\rangle\rangle_b : \{0,\ldots,b-1\}^* \to (0,1)$ as

$$\langle\langle w\rangle\rangle_b = \sum_{k=1}^{|w|} \frac{w_k}{b^k}.$$

to interpret a string of digits w as a base-b digital expansion (where the left-most symbol is the most significant bit) of the number $\langle\langle w\rangle\rangle_b$ in the unit interval. Such interpretations extend to languages so that

$$\langle\langle L\rangle\rangle_b = \{\langle\langle w\rangle\rangle_b : w \in L\}.$$

We fix the base as $b = 2$, and consider automata over the alphabet the two-dimensional boolean alphabet $\{0,1\} \times \{0,1\}$ to encode points in the plane. For example, Figure 5.11 depicts automata whose languages encode some of the regions occupied by tiles in the starting configuration shown in Figure 5.10a.



(a) $\mathcal{A}_U$ (b) $\mathcal{A}_V$

(c) $\mathcal{A}_W$

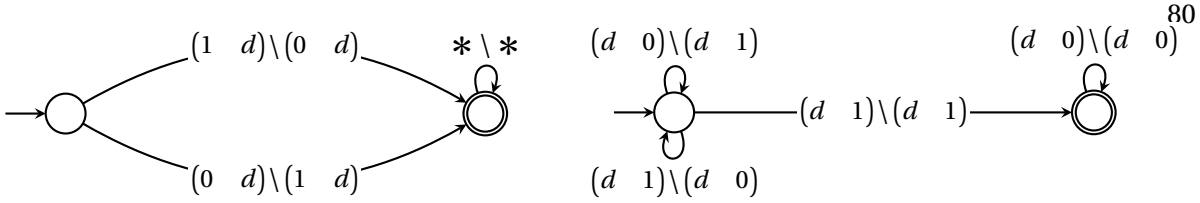
(d) $\mathcal{A}_Y$

Figure 5.11: Automata corresponding to some of the starting tiles shown in Figure 5.10.

Remark 5.1

The automata $\mathcal{A}_X$ and $\mathcal{A}_Z$ may be obtained by starting with $\mathcal{A}_W$ and $\mathcal{A}_Y$, respectively, and taking the logical complement of the x-coordinate on every non-looping transition and exchanging pairs of self loops on a common state labeled by $\begin{pmatrix}0 & 0\end{pmatrix}$ and $\begin{pmatrix}1 & 1\end{pmatrix}$ to ones labeled by $\begin{pmatrix}0 & 1\end{pmatrix}$ and $\begin{pmatrix}1 & 0\end{pmatrix}$.

We also design transducers implementing basic moves over the tiles, such as translations by $\frac{1}{2}$ and $\frac{1}{4}$ and reflection across the lines $x = \frac{1}{2}$ and $y = \frac{1}{2}$. Some of these transducers are depicted in Figure 5.12,



(a) Translate right by $\frac{1}{2}$

(b) Reflect about $y = \frac{1}{2}$

Figure 5.12: Transducers implementing some rigid transformations on the square $[0,1] \times [0,1]$.

in which an arbitrary digit is denoted by d and arbitrary pairs of digits are denoted by $*$.

The agent's objective is to apply these basic transformations to move each shape from its initial position into the goal region. To reduce the number of actions, the agent selects transformations for one of the shapes at a time and uses a special "submit" action to move to the next shape. We treat the collection of automata as a single NFA for the agent to view simultaneously, and specially mark the alphabet of the active automaton in the collection. Rewards are proportional to the overlap with the remaining exposed target shape when the submit action is used. On all other steps, a reward of -0.01 is given to encourage promptness.

The hyperparameters we used for PPO in this case study were 256 steps per update, 10 optimization epochs, a batch size of 64, a clip range $\epsilon = 0.1$, and a discount factor $\lambda = 0.99$. The resulting reward curve is shown in Figure 5.13. In this figure, the dark line shows the mean, and the shaded region shows the 10^{th} and 90^{th} percentiles over 5 training runs. The four annotated parts of the curve show the agent's progress. We selected an arbitrary training run, and executed the policy of the agent at these points in the training run to produce the configurations displayed in the figure's annotations.

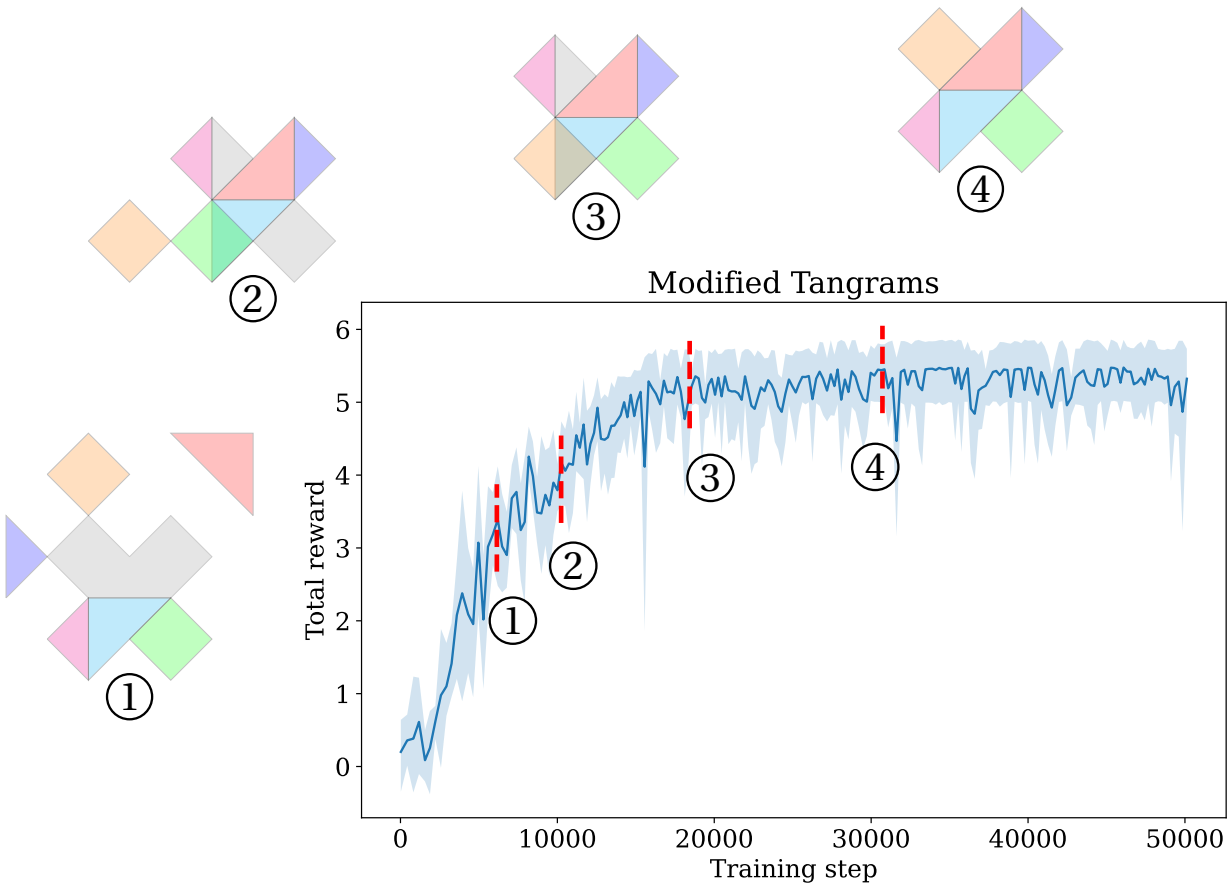


Figure 5.13: Annotated reward curve for modified tangrams.

Part II

Objectives Inspired by Regular Transformations

Chapter 6

Register Reward Machines

In this chapter, we propose the model of *register reward machines* as a generalization of the reward machine model of [Ica22]. Rather than starting with base model of finite-state transducers, we begin with the base model of *cost register automata* (CRA). CRA are a variation on *streaming string-to-tree transducers* [Alu+13], which generalize SSTs by allowing the machine to build nested hierarchical structures through register manipulation. Streaming string-to-tree transducers are a special case of *string tree transducers* [AD12; AD17], a generalization of SSTs that has trees as inputs and outputs, for which only outputs are trees while inputs are restricted to be strings. One can think of a CRA as a streaming string-to-tree transducer with a built-in interpretation, and, dually, one can think of a streaming string-to-tree transducer as an uninterpreted CRA.

6.1 Cost Register Automata

Before defining the model itself, we must introduce some notation and terminology.

Let F be a set of function symbols, each with a non-negative fixed arity (*i.e.* input dimension) and single output dimension. Assume that in F there finitely many functions of arity greater than zero and that there are infinitely many constants (*i.e.* function symbols of arity-0). Let $\textsc{term}(F)$ denote the set of all well-formed terms over F. A *cost grammar* defines a regular subset of $\textsc{term}(F)$, *i.e.* the set of all parse trees for these terms is a regular tree language [Tha67]. For a cost grammar G over F, let $\textsc{term}(G)$ represent the regular subset of $\textsc{term}(F)$ generated by G.

A *cost model* is a triple $\langle \mathbb{D}, G, [\![\cdot]\!] \rangle$, where $\mathbb{D}$ is an arbitrary set called the *cost domain*, G is a cost

grammar, and $[\![\cdot]\!] : \textsc{term}(F) \to \mathbb{D}$ is an *interpretation function* such that $[\![c]\!] \in \mathbb{D}$ for all constants $c \in F$ and $[\![f(c_1, ..., c_n)]\!] : \mathbb{D}$ for all n-ary functions $f \in F$. Given a set of variables X, define the set of expressions $\textsc{expr}_G(X)$ to be the extension of $\textsc{term}(G)$ in which leaf nodes may be elements of X. We extend the interpretation function so that each expression e is interpreted as an m-ary function $[\![e]\!] : \mathbb{D}^m \to \mathbb{D}$, where $m = |X|$.

A *valuation* is a function $\mathcal{V} : X \to \mathbb{D}$ that maps each variable to an element of the cost domain. An expression $e \in \textsc{expr}_G(X)$ and a valuation $\mathcal{V}$ may be combined to obtain an element of the cost domain as[1]

$$[\![e]\!](\mathcal{V}(x_1), ..., \mathcal{V}(x_m)).$$

Now, we are prepared to define the model.

Definition 6.1 – Cost Register Auomtaton (CRA)

Syntax: A cost register automaton is specified by a tuple $\langle \mathbb{C}, \Sigma, Q, q_0, X, \mathcal{V}_0, \delta, \rho, \varphi \rangle$, where

- $\mathbb{C} = \langle \mathbb{D}, G, [\![\cdot]\!] \rangle$ is a cost model;
- Σ is a finite alphabet;
- Q is a finite set of states;
- $q_0 \in Q$ is a distinguished initial state;
- X is a finite set of registers;
- $\mathcal{V}_0 : X \to \mathbb{D}$ is an initial valuation that maps each registers to an element of the cost domain;
- $\delta : Q \times \Sigma \to Q$ is a state transition function;
- $\rho : Q \times \Sigma \times X \to \textsc{expr}_G(X)$ is a register update function;
- $\varphi : Q \to \textsc{expr}_G(X)$ is an output function.

Semantics: Extend the transition function to $\delta^* : \Sigma^* \to Q$ such that $\delta^*(\varepsilon) = q_0$ and $\delta^*(w\sigma) = \delta(\delta^*(w), \sigma)$. Processing a word w results in a change of valuation from $\mathcal{V}_0$ to $\mathcal{V}_w$ according to the rules

$$\mathcal{V}_\varepsilon(x) = \mathcal{V}_0(x) \qquad \text{and} \qquad \mathcal{V}_{w\sigma}(x) = [\![\rho(\delta^*(w), \sigma, x)]\!](\mathcal{V}_w(X)).$$

The semantics of a CRA $\mathcal{A}$ take the form of a function $[\![\mathcal{A}]\!] : \Sigma^* \to \mathbb{D}$, defined by the equation

$$[\![\mathcal{A}]\!](w) = [\![\varphi \circ \delta^*(w)]\!](\mathcal{V}_w(X)).$$

As mentioned in Chapter 1, a function that can be represented by a *copyless* CRA is called a *regular cost function* .

[1] We write $[\![e]\!](\mathcal{V}(X))$ as shorthand for $[\![e]\!](\mathcal{V}(x_1), ..., \mathcal{V}(x_m))$.

6.2 Discounted Optimization & Register Reward Machines

With a few tweaks to Definition 6.1, we obtain a reasonable definition of register reward machines

.

Definition 6.2 – Register Reward Machine (RRM)

A register reward machine H is a CRA $\langle \mathbb{C}, \Sigma, Q, q_0, X, \mathcal{V}_0, \delta, \rho, \varphi \rangle$, where

- the cost domain $\mathbb{D}$ of the cost model $\mathbb{C} = \langle \mathbb{D}, G, [\![\cdot]\!] \rangle$ is some subset of $\mathbb{R}$ and G is based on operations that are well-defined this domain;
- the alphabet Σ is interpreted as a finite set of labels for some TMDP labeling function;
- the output $\varphi(q)$ is defined for every state $q \in Q$.

We now consider problems related to optimizing an infinite discounted sum of rewards generated by an RRM. Suppose we have a TMDP $T = \langle S, s_I, A, P, \Sigma, \ell, R \rangle$. Differing from Definition 4.1, we use S to represent TMDP states rather than X to avoid notational conflict with the set of registers. Additionally, TMDPs in this chapter are assumed to have deterministic reward functions $R : \Sigma^* \to \mathbb{R}$ rather than stochastic ones. Further, suppose that $H = \langle \mathbb{C}, \Sigma, Q, q_0, X, \mathcal{V}_0, \delta, \rho, \varphi \rangle$ is an RRM. The RRM H encodes the reward function R if, for every trajectory $\mathfrak{t} \in \mathrm{Traj}(T)$ with corresponding label sequence $w \in \Sigma^\omega$, the equation

$$R(w_{[1,n]}) = [\![H]\!](w_{[1,n]})$$

holds for all $n \in \mathbb{N}$. Then, the discounted payoff, with discount factor $\lambda \in [0,1)$, is

$$\sum_{n=1}^{\infty} r_n \lambda^{n-1} \quad = \quad \sum_{n=1}^{\infty} R(w_{[1,n]}) \lambda^{n-1} \quad = \quad \sum_{n=1}^{\infty} [\![H]\!](w_{[1,n]}) \lambda^{n-1}.$$

Technical Challenges

A crucial distinction between reward machines as they exist in the literature and RRMs is related to the number of possible reward values making up an infinite reward sequence. Standard reward machines associate a reward value to each of their states (or edges), and this restricts the reward values to elements of some finite subset of $\mathbb{R}$. In contrast, RRMs do not necessarily impose such a restriction, and reward values may come from some infinite subset of $\mathbb{R}$. This can be witnessed by even the most

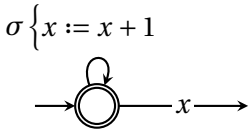


Figure 6.1: A simple register reward machine generating unbounded reward sequences.

simple RRMs such as those over the cost domain $\langle \mathbb{N}, +1 \rangle$, which store natural numbers in their registers and are allowed only the single operation of incrementing the values in their registers. Indeed, the RRM in Figure 6.1 has only a single state and a single register, yet produces unbounded reward sequences.

The infinitude of possible reward values poses several challenges that must be overcome in order to use RRMs for the purposes of sequential optimization.

(1) When considering unbounded reward sequences, we can no longer assume that discounted summations converge.

(2) Even if we can assure that reward values are coming from a bounded subset of $\mathbb{R}$—which allows us to assume convergence of discounted sums without issue—, we cannot assume that this bounded subset is finite.

As will be shown in the remainder of this chapter, the problem of unboundedness of reward sequences can be addressed at the level of the discounted objective function. The issues caused by infinitely many reward values, on the other hand, is a more difficult obstruction. To understand why, recall that the crucial operation used to enable optimization in environments with reward signals defined by standard reward machines is what we call the *product trick*. The product trick involves taking a product $T \otimes H$ of the environment T and a reward machine H and producing an MDP as a result. Assuming that T is finite, then M is finite also, since H is finite by definition. At this point, one can execute their favorite optimization algorithm for finite MDPs on M. If H is an RRM, however, the product trick is no longer a useful option, because the resulting decision process will not have a Markovian reward function in general. Consequently, it is not possible to apply off-the-shelf sequential optimization algorithms to solve optimization problems involving reward signals modeled by RRMs. To tackle the general situation, substantially different approaches seem to be required, which will take

the form of either (1) new algorithms for optimizing over finite environments with non-Markovian reward functions with infinite ranges, or (2) more intricate or clever variations on the idea of the product trick. Another option involves searching for subclasses of RRMs that generate reward sequences tame enough for the existing techniques to work with minor tweaks. The latter option of dealing with special cases appears easier than handling the general problem, but leaves us with a lot of different types of RRMs to sift through. In any case, we leave this problem open and return to addressing the problem of unboundedness of rewards.

The following result elucidates a relationship between growth rates of reward sequences and magnitudes of discount factors that is useful for determining when a discounted optimization problem is well-posed when rewards are unbounded. The proof relies on standard tools from the theory generating functions and power series [Wil90].

Proposition 6.1 – Reward Growth & Discounting

Suppose that $(r_n)_{n=1}^{\infty}$ is an unbounded sequence of real numbers.

(1) If $r_n = \text{poly}(n)$, then the λ-discounted sum is finite for all discount factors $\lambda < 1$.

(2) If $r_n = O(2^n)$, then there exists a constant $C \in (0,1]$ such that the λ-discounted sum is finite, for every discount factor $\lambda \in [0, C)$.

(3) If $r_n = \Omega\big(2^{n \log n}\big)$, there exists no $\lambda > 0$ such that the λ-discounted sum is finite.

Proof. Consider $F : [0,1] \to \mathbb{R} \cup \{\infty\}$ to be the ordinary generating function of an arbitrary reward sequence:

$$F(\lambda) = \sum_{n=1}^{\infty} r_n \lambda^{n-1}.$$

The radius of convergence of F is a non-negative real number (or ∞) C such that $F(\lambda) < \infty$ for all $\lambda \in [0, C)$. It can be expressed in terms of the reward sequence as

$$C = \limsup_{n \to \infty} \frac{1}{\sqrt[n]{r_n}},$$

where, by convention, $\frac{1}{0} = \infty$.

(1) To see that the first item of the claim holds, recall that $\text{poly}(n) = O\big(2^{\log n}\big)$. The radius of conver-

gence when $r_n = \mathsf{poly}(n)$ is, for some constants $c, d > 0$, given by

$$C \quad = \quad \limsup_{n\to\infty} \frac{1}{\sqrt[n]{c^{d\log n}}} \quad = \quad 1,$$

since $\dfrac{\log n}{n} \to 0$ as $n \to \infty$.

(2) The second item follows from the fact that that, for any constants $c, d > 0$,

$$C \quad = \quad \limsup_{n\to\infty} \frac{1}{\sqrt[n]{c^{dn}}} \quad = \quad \frac{1}{c^d}.$$

(3) To prove the third claim, notice that if the exponent in the denominator of the above equation grows with n, however slowly, then the radius vanishes due to the numerator being constant. That is, for any constants $c, d > 0$,

$$C \quad = \quad \limsup_{n\to\infty} \frac{1}{\sqrt[n]{c^{dn\log n}}} \quad = \quad \limsup_{n\to\infty} \frac{1}{c^{d\log n}} \quad = \quad 0. \qquad \square$$

Proposition 6.1 tells us that the fastest reward growth under which discounted optimization can possibly be carried out is on the order of $O(2^n)$, but this already rules out the use of discount factors close to 1. In light of Proposition 6.1, we would like to understand nature of the growth rates of various types of RRMs. Ideally, we would like to identify RRMs that produce reward sequences with polynomially bounded growth.

6.2.1 Polynomial Register Updates

A *polynomial* RRM is one allowed to update its register by applying polynomial functions to them. More precisely, these are RRMs that can perform assignments of the form $x := e$, where e is an expression captured by the grammar

$$e := ee \mid e + e \mid x \mid c.$$

Fixing the cost domain as $\mathbb{Q}$, such expressions correspond to elements of the ring of polynomials $\mathbb{Q}[X]$ with variables identified with those of some CRA.

Proposition 6.2 – Reward Growth Of Polynomial Register Updates

If each register update in a polynomial RRM corresponds to a polynomial of degree at least 2, then each reward sequence it generates grows as

$$r_n = \Omega\left(2^{2^n}\right).$$

Proof. Suppose that H is a polynomial RRM $\langle \mathbb{C}, \Sigma, Q, q_0, X, \mathcal{V}_0, \delta, \rho, \varphi \rangle$ with m registers. The valuation $\mathcal{V}_n(x)$ of a register x at time n equal to the value $P(\mathcal{V}_0(X))$, where P is a polynomial obtained as a composition of polynomials $P_1 \circ ... \circ P_n$, each corresponding to a register update $\rho\left(q_{k-1}, \sigma_k, x_{i_k,k}\right)$ applied along the current run. More precisely,

$$P_k = \left[\!\left[\rho\left(q_{k-1}, \sigma_k, x_{i_k}\right)\right]\!\right],$$

where x_{i_k} is the variable x_i indexed by k, and so the value held in x at time n is

$$\mathcal{V}_n(x) \quad = \quad \left[\!\left[\rho\left(q_{n-1}, \sigma_n, x_{i_n}\right)\right]\!\right](\mathcal{V}_{n-1}(X)) \quad = \quad P_n \circ \mathcal{V}_{n-1}(X).$$

From here, it follows inductively that

$$\mathcal{V}_n(x) \quad = \quad P_n \circ ... \circ P_1(\mathcal{V}_0(X)) \quad = \quad P(\mathcal{V}_0(X)).$$

The claim can now be proven using the fact that degrees of polynomials multiply under composition [LN73]: for polynomials P_1 and P_2 defined over a field[2] ,

$$\deg(P_1 \circ P_2) = \deg(P_1)\deg(P_2).$$

If $\deg(P_k) \geq 2$, for each $k \in \{1, ..., n\}$, then it follows that $\deg(P) \geq 2^n$. This implies that

$$\mathcal{V}_n(x) = \Omega\left(c^{2^n}\right),$$

where we let $\max_{x \in X} \mathcal{V}_0(x) = c$, as it is constant with respect to n. Since the output function $\varphi(q_n)$ is a polynomial,

$$r_n \quad = \quad [\![\varphi(q_n)]\!](\mathcal{V}_n(X)) \quad = \quad \Omega\left(c^{2^n}\right) \quad = \quad \Omega\left(2^{2^n}\right). \qquad \square$$

[2] Note that this equation holds with repsect to any field, not just $\mathbb{Q}$.

Proposition 6.2 establishes that polynomial RRMs are not well suited for the purposes of discounted optimization. While unfortunate, this result further supports the premise of this thesis, since the only polynomials that can be used for updates by copyless polynomial RRMs—given the grammar specified at the beginning of this subsection—are degree 1 polynomials. Proposition 6.2 reaffirms our commitment to regularity, and tells us that we should turn our attention back towards simpler kinds of RRMs. Heeding this message, we look towards simpler RRMs.

6.2.2 Additive Register Updates

Additive RRMs are those allowing only the operation of addition to their registers, so that register updates are of the form $x := e$ where e is expression according to the grammar

$$e := e + e \mid x \mid c.$$

A simple observation is that reward sequences are bounded linearly if, and only if, they are generated by *copyless* additive RRMs, such as the one displayed in Figure 6.1. On the other hand, it is not difficult to come up with copyful additive RRMs that generate reward sequences of exponential growth. The CRA depicted in Figure 2.4, for example, generates Fibonacci numbers.

Proposition 6.3 – Reward Growth Of Additive Register Updates

Suppose that H is an additive RRM.

(1) Any reward sequence of H exhibits growth at most linear or at least exponential:

$$r_n = O(n) \qquad \text{or} \qquad r_n = 2^{\Omega(n)}.$$

(2) There exists a copyless additive CRA equivalent to H if, and only if, $r_n = O(n)$ holds for all reward sequences of H.

With the context provided by Proposition 6.1, the dichotomy established by Proposition 6.3 exposes another link between regularity and optimization. As mentioned earlier, *regular cost functions* are modeled by *copyless* CRA, and these are precisely the CRA producing exclusively reward sequences of linear-bounded growth. The regular cost functions modeled by copyless additive CRA are known as *additive regular cost functions* [AR13]. Thus, additive RRMs producing exclusively reward sequences

of linear-bounded growth model *additive regular reward functions* in this same sense. In light of Proposition 6.1, copyless additive RRMs are fully compatible with discounted optimization—as far as reward growth is concerned, that is—, whereas copyful additive RRMs are only partially compatible with the discounted payoff.

Since it would be difficult to come up with simpler cost domains than those of additive RRMs, we now turn to a more complex cost model.

6.2.3 Affine Register Updates & Discounting The Past

Affine RRMs are permitted to update their registers using both addition and multiplication by constants according to the grammar

$$e := e + e \mid de \mid x \mid c.$$

Of particular interest, is a subclass of affine RRMs called *past-discounting* RRMs characterized by restricting the above grammar to

$$e := de + c \mid x,$$

and restricting any constant d to be from $\mathbb{Q} \cap [0,1]$. Notice that this grammar implicitly enforces the copyless restriction, making discounting RRMs quite regular.

Past-discounting RRMs get there name from the fact that, if only a single multiplicative constant γ is available, then a sequence of n updates performed by such a machine results in a value captured by a sum of the form

$$\sum_{k=1}^{n} \gamma^{n-k} r_k. \qquad \text{(Past-Discounted Sum)}$$

This kind of sum is called a *past-discounted sum*, because it follows the pattern of a standard discounted sum $\sum_{k=1}^{n} r_k \lambda^{k-1}$, but in reverse order. The most recent additions to a past-discounted sum contribute the most to the overall sum, since the existing sum is scaled by a number lower than 1 at every update. This mirrors the standard discounted sum, to which the initial summands are never scaled and therefore contribute the most to its overall value. If a decision processes has a reward function definable by a past-discounting RRM, then optimization problems involve minimizing or maximizing payoff

functions evaluated with respect to reward sequences in which r_n corresponds to a past discounted sum $\sum_{k=1}^{n} \gamma^{n-k} r_k$. The next two chapters introduce and study *past-discounted* optimization problems involving such reward sequences, independently of the notion of RRMs.

Chapter 7

A Theory of Depreciating Assets

Time preference [FLO02; LJ92] refers to the tendency of rational agents to value potential outcomes in proportion to the amount of time until such an outcome may be realized. In other words, agents prefer to receive rewards sooner rather than later, and, symmetrically, agents prefer to experience negative outcomes later rather than sooner. This idea is often codified mathematically in terms of *discounting* and has been applied widely in areas such as game theory [FV96; NS03; Sha53], economics [Hea07; Phi99], control theory [FS01; Put94], and reinforcement learning [SB98].

Following the pioneering work of Shapley [Sha53], we consider *exponential discounting* in which agents generate an infinite sequence of rewards $(r_n)_{n\in\mathbb{N}}$. Their goal is to optimize a discounted sum

$$\sum_{n=1}^{\infty} \lambda^{n-1} r_n, \qquad \text{(Discounted Sum of Rewards)}$$

where $\lambda \in [0,1)$ is a *discount factor* that quantifies the magnitude of time preference. With discount factors close to 0, the initial terms of the series contribute disproportionately to the limit to which the discounted sum converges. On the other hand, for discount factors close to 1, terms occurring later in the series become more significant. Note that the discounted optimization problem is equivalent to the problem of optimizing expected terminal payoff of the process given a constant probability $(1-\lambda)$ of terminating operations at any point.

A notable characteristic of discounting is an implicit assumption that rewards are static: the scaled reward $\lambda^{n-1} r_n$ encompasses the totality of the value gained by taking a particular action at time n. Nothing that occurs after time n can change the fact that the action taken at time n contributes

$\lambda^{n-1} r_n$ to the overall discounted sum. It is not too difficult, however, to find scenarios where such an assumption is inappropriate. Consider, for instance, the most basic and ubiquitous of rewards used to incentivize human behaviors: money. Assets in the form of liquid currency or cash lose value with time due to inflation. More generally, our work focuses on situations where the worth of assets decrease with time, which is quite common. This pattern is found in problems of inventory management for businesses that deal with goods that are perishable or with tools and machinery whose values decay with age and use.

Depreciation , a term borrowed from the fields of finance [Hot25; Tay23], accounting [Bur72; Wri64], and economics [HW80; HW96; Pre38], describes exactly the phenomenon where economic worth decays with time. We propose a notion of *exponential depreciation* that is inspired by exponential discounting and is based on applying the same basic principle of time preference, but to an agent's past rather than its future. Mathematically, this amounts to evaluating payoffs, not directly on a reward sequence $(r_n)_{n\in\mathbb{N}}$, but on a *depreciating asset sequence*

$$\left(\sum_{k=1}^{n} r_k \gamma^{n-k}\right)_{n\in\mathbb{N}} \qquad \text{(Depreciating Asset Sequence)}$$

where $\gamma \in [0,1]$ is a *depreciation factor* that quantifies the rate of depreciation.

The Declining Balance Depreciation Method

Let us take a moment to expound upon the connection between exponential depreciation and a similar notion used in finance, accounting, and economics. In these fields, depreciation may be articulated as follows: an asset depreciates if its so-called *book value* (*i.e.* its worth) decreases with time. At the time of acquisition (time 0), the book value B_0 of an asset is set as the price paid to acquire it. The *depreciation expense* of the asset over period n is then defined by the difference

$$D_n = B_{n-1} - B_n.$$

The precise methods used to determine depreciation expenses and book values vary depending on the specifics of a given situation. Some of the more popular methods include "*straight line*" depreciation, "*sum-of-years-digits*" depreciation, and "*declining balance*" depreciation [GMP16].

It turns out that our method of depreciation is intimately connected to the declining balance method. Given a parameter $\delta \in [0,1]$, the declining balance method sets the depreciation expense in terms of the previous book value:

$$D_n = \delta B_{n-1},$$

for a given time n. Using the defining equation of the depreciation expense, a recurrence

$$B_n = (1-\delta)B_{n-1}$$

is obtained describing the evolution of the book value, without explicit reference to the depreciation expense.

The connection to our notion of exponential depreciation is made apparent by reconstructing the declining balance method, this time giving precedence to the book value rather than the depreciation expense. Given a depreciation factor $\gamma \in [0,1]$, we directly define a recurrence for the book value

$$B_n = \gamma B_{n-1}.$$

Then one may obtain the depreciation expenses according to the derived equation

$$D_n = (1-\gamma)B_{n-1}.$$

As far as we could find, finite sums of the form $\sum_{k=1}^{n} r_k \gamma^{n-k}$ appear in only two veins of relevant work[1] . As suggested by the preceding discussion, one of those veins is found in the finance literature related to the declining balance method [Bur72; GMP16]. The other is found in the literature around cost register automata [Alu+13], as discussed in Chapter 6.

Optimizing Depreciating Asset Streams

We investigate optimization of both discounted values and long-run average values of depreciating asset streams. The inquiry is carried out in the setting of two-player zero-sum stochastic games, as

[1] To the best of our knowledge, limits of expressions involving these sums have not previously been studied.

introduced by [Sha53]. Our approach involves incorporating depreciation dynamics into the definitions of these payoffs. More concretely, we consider families of payoffs based on discounted sums of depreciating asset streams

$$\sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=1}^{n} r_k \gamma^{n-k} \qquad \text{(Discounted Sum of Depreciating Assets)}$$

and long-run averages[2] of depreciating asset streams

$$\sum_{n=1}^{\infty} \frac{1}{n} \sum_{k=1}^{n} r_k \gamma^{n-k}. \qquad \text{(Long-Run Average of Depreciating Assets)}$$

Our choice of optimization criteria is motivated by the central roles played by discounted and average payoffs in optimal control and game theory. A multitude of deep results exist about these objectives, characterizing their values, analyzing their strategic complexity, and elucidating the relationship between them in the context of stochastic games and subclasses thereof [BK76; BK78; CDS11; CM12; FS01; FV96; FTV02; Gil58; LL69; MP70; MN81; Mil02; Oli14; Put94; Sha53; SHR13; Zil16a; Zil16b; Zil18]. Additionally, there is an extensive body of work focussed on developing algorithms [ACM12; Bor+11; Bor+17; Bor+18; Han+11; HK66; Oli21; RF91; RS03; RCN73] for computing and approximating optimal values and strategies for these payoffs and establishing bounds on the computational complexities of such methods [AM09; Bor+13; Bor+19; CMH08; CI14; EY10; Hal07; HI13; HMZ13; Sid+20].

The following two subsections are intended to provide, in a non-rigorous way, some intuition towards the gist of the key mathematical insights underpinning the reductions of Theorems 7.1 and 7.2. Both of them consider a periodic sequence of rewards $(r_n)_{n\in\mathbb{N}} = 3,4,5,3,4,5,\ldots$ and the corresponding γ-depreciating asset sequence

$$3, \quad (3\gamma+4), \quad (3\gamma^2+4\gamma+5), \quad (3\gamma^3+4\gamma^2+5\gamma+3), \quad (3\gamma^4+4\gamma^3+5\gamma^2+3\gamma+4), \ldots$$

Discounting Over Depreciating Asset Streams

The λ-discounted value of the γ-depreciating asset sequence of $(r_n)_{n\in\mathbb{N}}$ can be obtained heuristically via the following informal derivation. The first several steps are simply elementary algebraic

[2] These series do not always converge, so the long-run average optimization objective is strictly stated in terms of either the liminf or the limsup of the sequence of partial sums. As can be seen in Definition 7.2, our convention is to use liminf.

manipulations that are carried out without regard for matters of convergence. The final two lines are obtained via the basic property $\sum_{n=1}^{\infty} z^{n-1} = \frac{1}{1-z}$ of geometric series with $z \in [0,1)$.

$$\sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=1}^{n} r_k \gamma^{n-k}$$

$$=$$

$$3 + \lambda(3\gamma + 4) + \lambda^2(3\gamma^2 + 4\gamma + 5) + \lambda^3(3\gamma^3 + 4\gamma^2 + 5\gamma + 3) + \lambda^4(3\gamma^4 + 4\gamma^3 + 5\gamma^2 + 3\gamma + 4) + \ldots$$

$$=$$

$$(3 + 3\lambda\gamma + 3\gamma^2\lambda^2 + \ldots) + (4\lambda + 4\lambda^2\gamma + 4\lambda^3\gamma^2 + \ldots) + (5\lambda^2 + 5\lambda^3\gamma + \lambda^5\gamma^2 + \ldots) + (3\lambda^3 + 3\lambda\gamma^4 + \ldots) + \ldots$$

$$=$$

$$3(1 + \lambda\gamma + \gamma^2\lambda^2 + \ldots) + 4\lambda(1 + \lambda\gamma + \lambda^2\gamma^2 + \ldots) + 5\lambda^2(1 + \lambda\gamma + \lambda^2\gamma^2 + \ldots) + 3\lambda^3(1 + \lambda\gamma + \gamma^2\lambda^2 + \ldots) + \ldots$$

$$=$$

$$\frac{1}{(1-\lambda\gamma)}(3 + 4\lambda + 5\lambda^2 + 3\lambda^3 + \ldots)$$

$$=$$

$$\frac{3 + 4\lambda + 5\lambda^2}{(1-\lambda\gamma)(1-\lambda^3)}.$$

Notice that the final line consists of a product in which one term is the λ-discounted value of $(r_n)_{n\in\mathbb{N}}$. This suggests that the discounted value of a depreciating asset sequence is equal to the product of $\frac{1}{1-\lambda\gamma}$ and the discounted value of the original asset sequence. We confirm that this is not a mere coincidence, and establish that the relationship holds generally.

Averaging Over Depreciating Asset Streams

The sequence of partial sums for the long-run average of the γ-depreciating asset sequence $\left(\sum_{k=1}^{n} r_k \gamma^{n-k}\right)_{n\in\mathbb{N}}$ of a reward sequence $(r_n)_{n\in\mathbb{N}}$ is

$$3, \quad \frac{3\gamma + 4}{2}, \quad \frac{3\gamma^2 + 4\gamma + 5}{3}, \quad \frac{3\gamma^3 + 4\gamma^2 + 5\gamma + 3}{4}, \quad \frac{3\gamma^4 + 4\gamma^3 + 5\gamma^2 + 3\gamma + 4}{5}, \ldots$$

Based on classical Tauberian results [BK76], it is tempting to conjecture that the λ-discounted sum of the γ-depreciating asset sequence converges to this mean as $\lambda \to 1$. That is, taking $\lambda \to 1$ from below,

the discounted depreciating value could be expected to approach the long-run average depreciating value. The following derivation illustrates the mathematical intuition motivating this conjecture.

$$\begin{aligned}
\sum_{n=1}^{\infty} \frac{1}{n} \sum_{k=1}^{n} r_k \gamma^{n-k} &= \lim_{\lambda \to 1} \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=1}^{n} r_k \gamma^{n-k} \\
&= \lim_{\lambda \to 1} (1-\lambda) \frac{3 + 4\lambda + 5\lambda^2}{(1-\lambda\gamma)(1-\lambda^3)} \\
&= \lim_{\lambda \to 1} \frac{3 + 4\lambda + 5\lambda^2}{(1-\lambda\gamma)(1+\lambda+\lambda^2)} \\
&= \frac{4}{1-\gamma} \\
&= \frac{1}{1-\gamma} \sum_{n=1}^{\infty} \frac{r_n}{n}
\end{aligned}$$

Sure enough, we prove that such a relationship holds in general.

7.1 Discounting Over Depreciating Asset Streams

We now study discounted optimization under depreciating asset dynamics.

Definition 7.1 – Discounted Depreciating Payoff

Let $\Re = (r_n)_{n \in \mathbb{N}}$ be a bounded reward. The discounted depreciating payoff $\mathsf{D}_\lambda^\gamma : \mathbb{R}^\omega \to \mathbb{R}$, with discount factor $\lambda \in [0,1)$ and depreciation factor $\gamma \in [0,1]$, is captured by the equation

$$\mathsf{D}_\lambda^\gamma(\mathfrak{r}) = \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=1}^{n} r_k \gamma^{n-k}. \qquad \text{(Discounted Depreciating Payoff)}$$

The corresponding value function $V_\lambda^\gamma : X \to \mathbb{R}$ is defined, with respect to the state space X of an arbitrary MDP M, as

$$V_\lambda^\gamma(x) = \sup_{\pi \in \Pi_M} \mathbb{E}_x^\pi \left[\sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=1}^{n} r_k \gamma^{n-k} \right]. \qquad \text{(Discounted Depreciating Value)}$$

Our main result establishes a reduction from the discounted depreciating value to the standard discounted value.

Theorem 7.1 – From Discounted Depreciating Values To Discounted Values

Suppose that x is a state of an arbitrary Markov decision process. For any discount factor $\lambda \in [0,1)$ and any depreciation factor $\gamma \in [0,1]$,

$$V_\lambda^\gamma(x) = \frac{V_\lambda(x)}{1-\lambda\gamma}.$$

Proof. Consider an arbitrary with reward sequence $\mathfrak{r} = (r_n)_{n\in\mathbb{N}}$. By splitting the term λ^{n-1} occurring in the definition of $\mathsf{D}_\lambda^\gamma(\mathfrak{r})$ into the product $\lambda^{n-k}\lambda^{k-1}$ and distributing these factors into the inner summation, we obtain the expression

$$\sum_{n=1}^{\infty}\sum_{k=1}^{n} \lambda^{k-1} r_k \lambda^{n-k}\gamma^{n-k}. \tag{7.1}$$

The next step of the proof relies on the following classical result of real analysis [Rud76, Theorem 3.50].

Mertens' Theorem. *Let $\sum_{n=1}^{\infty} c_n = C$ and $\sum_{n=1}^{\infty} d_n = D$ be two convergent series of real numbers. If at least one of the series converges absolutely, then their Cauchy product converges to the product of their limits:*

$$\left(\sum_{n=1}^{\infty} c_n\right)\left(\sum_{n=1}^{\infty} d_n\right) = \sum_{n=1}^{\infty}\sum_{k=1}^{n} c_k d_{n-k} = CD.$$

The series (7.1) may be factored into the Cauchy product

$$\left(\sum_{n=1}^{\infty}(\lambda\gamma)^{n-1}\right)\left(\sum_{n=1}^{\infty}\lambda^{n-1}r_n\right), \tag{7.2}$$

and since both terms in this Cauchy product converge absolutely, Mertens' theorem applies. Thus, noticing that the left-hand series is geometric, the expression (7.2) is equivalent to

$$\frac{1}{1-\lambda\gamma}\sum_{n=1}^{\infty}\lambda^{n-1}r_n.$$

Since this series is exactly the definition of $\mathsf{D}_\lambda(\mathfrak{r})$, it follows that the identity

$$\mathsf{D}_\lambda^\gamma(\mathfrak{r}) = \frac{\mathsf{D}_\lambda(\mathfrak{r})}{1-\lambda\gamma}$$

holds for every possible reward sequence $\mathfrak{r}$. In turn, this implies

$$V_\lambda^\gamma(x) = \mathrm{Val}\left(\frac{\mathsf{D}_\lambda}{1-\lambda\gamma}, x\right).$$

Finally, using linearity of expectation and independence with respect to the infimum/supremum in the definition of the values, $\frac{1}{1-\lambda\gamma}$ may be factored out from the right-hand side of this equation to yield

$$V_\lambda^\gamma(x) = \frac{V_\lambda(x)}{1-\lambda\gamma}.$$ □

Two immediate consequences of Theorem 7.1 are characterizations of the strategic complexity and the computational complexity of discounted depreciating optimization.

Corollary 7.1 – Strategic Complexity of Discounted Depreciating Optimization

For any discounted depreciating payoff over any finite MDP, there exists an optimal positional policy.

Recall that the value problem, with respect to an MDP M and an objective F, asks to determine, for some state x and bound $b \in \mathbb{Q}$, whether $b \leq \mathrm{Val}_M(\mathsf{F}, x)$.

Corollary 7.2 – Computational Complexity of Discounted Depreciating Optimization

Over any finite Markov decision process

(1) the value problem for any discounted depreciating payoff is LOGSPACE-reducible to the value problem for a discounted payoff;

(2) values and optimal policies under discounted depreciating payoffs are computable in polynomial time.

Another corollary of Theorem 7.1 is the following.

Corollary 7.3 – Discounted Depreciating Optimality Equations

Suppose that $M = \langle X, A, P, R\rangle$ The discounted depreciating value is the unique solution of the system of equations

$$V_\lambda^\gamma(x) = \max_{a\in A} \frac{R(x,a)}{1-\lambda\gamma} + \lambda \sum_{y\in X} P(y \mid x, a) V_\lambda^\gamma(y). \qquad (x \in X)$$

Coroallary 7.3 allows the formulation of an associated Q-value as the unique solution to the system of equations

$$Q_\lambda^\gamma(x,a) = \frac{R(x,a)}{1-\lambda\gamma} + \lambda \sum_{y\in X} P(y \mid x, a) \max_{b\in A} Q_\lambda^\gamma(y,b) \qquad (x \in X)$$

This forms a basis for an adaptation of Q-learning for discounted depreciating payoffs, which

iteratively approximates the Q-value along a sample trajectory by iterating the assignment

$$Q_{n+1}(x_n, a_n) := (1-\alpha_n)Q_n(x_n, a_n) + \alpha_n\left(\frac{r_n}{1-\lambda\gamma} + \lambda \max_{b\in A} Q_n(x_{n+1}, b)\right), \tag{7.3}$$

where $(\alpha_n)_{n\in\mathbb{N}}$ is a sequence of learning rates such that $\alpha_n \in (0,1)$ for all $n \in \mathbb{N}$.

Corollary 7.4 – Convergent Q-Learning For Discounted Depreciating Payoffs

If each state-action pair of a finite MDP is encountered infinitely often and the learning rates satisfy the Robbins-Monroe convergence criteria $\sum_{n=0}^{\infty} \alpha_n = \infty$ and $\sum_{n=0}^{\infty} \alpha_n^2 < \infty$, then iterating (7.3) converges almost surely in the limit to the discounted depreciating Q-value:

$$Q_n \xrightarrow{\text{a.x.}} Q_\lambda^\gamma \qquad \text{as} \qquad n \to \infty.$$

7.2 Averaging Over Depreciating Asset Streams

Let us now consider the asymptotic average evaluation criterion, given that assets depreciate.

Definition 7.2 – Average Depreciating Payoff & Value

Let $\Re = (r_n)_{n\in\mathbb{N}}$ be a bounded reward sequence. The average depreciating payoff $\mathsf{D}^\gamma : \mathbb{R}^\omega \to \mathbb{R}$, for depreciation factor $\gamma \in [0,1]$, is defined

$$\mathsf{D}^\gamma(\mathfrak{r}) = \liminf_{n\to\infty} \sum_{k=1}^{n} \sum_{i=1}^{k} \frac{r_i \gamma^{k-i}}{n}. \qquad \text{(Average Depreciating Payoff)}$$

The average depreciating value function $V^\gamma : X \to \mathbb{R}$ is defined, with respect to the state space X of an arbitrary MDP M, by the equation

$$V^\gamma(x) = \sup_{\pi\in\Pi_M} \mathbb{E}_x^\pi \left[\liminf_{n\to\infty} \sum_{k=1}^{n} \sum_{i=1}^{k} \frac{r_i \gamma^{k-i}}{n}\right]. \qquad \text{(Average Depreciating Value)}$$

Much like in the previous section, the main result here establishes a reduction from average depreciating payoffs to average payoffs via an equational relationship between their respective values. First, however, we must state and prove the following pair of lemmas on which the proof of Theorem 7.2 relies.

Lemma 7.1

For any finite sequence $r_1, ..., r_n$ of real numbers, discount factor $\lambda \in [0,1)$, and depreciation factor $\gamma \in [0,1]$,

$$\sum_{k=1}^{n} \sum_{i=1}^{k} \frac{r_i \gamma^{k-i}}{n} = \sum_{k=1}^{n} \frac{r_k(1-\gamma^{n+1-k})}{n(1-\gamma)}. \tag{7.4}$$

Proof. We proceed by induction on n.

BASE CASE: $n = 1$.

If $n = 1$, then, it is clear that both sides of Equation (7.4) evaluate to r_1.

INDUCTIVE CASE: *If Equation (7.4) holds for $n-1$, then Equation (7.4) holds for n.*

Suppose that Equation (7.4) holds for $n-1$. By splitting the summation on the left-hand side of Equation (7.4), we obtain the expression

$$\sum_{k=1}^{n-1} \sum_{i=1}^{k} \frac{r_i \gamma^{k-i}}{n} + \sum_{k=1}^{n} \frac{r_k \gamma^{n-k}}{n}.$$

Factoring $\frac{n-1}{n}$ from the double summation yields

$$\frac{n-1}{n} \sum_{k=1}^{n-1} \sum_{i=1}^{k} \frac{r_i \gamma^{k-i}}{n-1} + \sum_{k=1}^{n} \frac{r_k \gamma^{n-k}}{n}.$$

Now, applying the inductive hypothesis, this may be rewritten as

$$\frac{n-1}{n} \sum_{k=1}^{n-1} \frac{r_k(1-\gamma^{n-k})}{(n-1)(1-\gamma)} + \sum_{k=1}^{n} \frac{r_k \gamma^{n-k}}{n}.$$

Factoring out $\frac{1}{n(1-\gamma)}$ from the entire expression, we get

$$\frac{1}{n(1-\gamma)} \left(\sum_{k=1}^{n-1} r_k(1-\gamma^{n-k}) + (1-\gamma) \sum_{k=1}^{n} r_k \gamma^{n-k} \right).$$

Distributing the terms r_k and $(1-\gamma)$ through the appropriate summations, we obtain

$$\frac{1}{n(1-\gamma)} \left(\sum_{k=1}^{n-1} \left(r_k - r_k \gamma^{n-k} \right) + \sum_{k=1}^{n} \left(r_k \gamma^{n-k} - r_k \gamma^{n+1-k} \right) \right).$$

Removing those terms that cancel additively yields

$$\frac{1}{n(1-\gamma)} \left(\sum_{k=1}^{n} r_k - \sum_{k=1}^{n} r_k \gamma^{n+1-k} \right).$$

Finally, we factor out r_k to consolidate the two summations into one and then distribute the leftmost term to obtain Equation (7.4):

$$\sum_{k=1}^{n} \frac{r_k(1-\gamma^{n+1-k})}{n(1-\gamma)}.$$

Therefore, we have proven that if Equation (7.4) holds for reward sequences of length $n-1$, then it also holds for reward sequences of length n. □

Lemma 7.2

For any bounded reward sequence $(r_n)_{n\in\mathbb{N}}$, discount factor $\lambda \in [0,1)$, and depreciation factor $\gamma \in [0,1]$,

$$\lim_{n\to\infty} \sum_{k=1}^{n} \frac{r_k \gamma^{n+1-k}}{n(1-\gamma)} = 0.$$

Proof. Factoring out the constant term in the denominator of the left-hand side of the claimed equation, we obtain the equivalent expression

$$\frac{1}{1-\gamma} \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} r_k \gamma^{n-k}. \tag{7.5}$$

Letting $r_\downarrow$ and $r_\uparrow$ be the respective lower and upper bounds of the sequence, namely

$$r_\downarrow = \min_{X\times A} R(x,a) \qquad \text{and} \qquad r_\uparrow = \max_{X\times A} R(x,a),$$

we can bound the finite sum from within the limit of (7.5), both above and below, as

$$\frac{r_\downarrow(1-\gamma^{n-1})}{1-\gamma} \quad \leq \quad \sum_{k=1}^{n} r_k \gamma^{n-k} \quad \leq \quad \frac{r_\uparrow(1-\gamma^{n-1})}{1-\gamma}.$$

Lastly, noticing

$$\lim_{n\to\infty} \frac{r_\downarrow(1-\gamma^{n-1})}{n(1-\gamma)} \quad = \quad 0 \quad = \quad \lim_{n\to\infty} \frac{r_\uparrow(1-\gamma^{n-1})}{n(1-\gamma)},$$

it follows that

$$0 = \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} r_k \gamma^{n-k}.$$

□

Now we are in position to state and prove the reduction.

Theorem 7.2 – From Average Depreciating Values To Average Values

Suppose that x is a state of an arbitrary Markov decision process. For any depreciation factor $\gamma \in [0,1]$,

$$V^{\gamma}(x) = \frac{V(x)}{1-\gamma}.$$

Proof. Applying Lemma 7.1, the average depreciating payoff may be rewritten as

$$\liminf_{n\to\infty} \sum_{k=1}^{n} \frac{r_k(1-\gamma^{n+1-k})}{n(1-\gamma)}.$$

Distributing the product in the numerator and then splitting the summation into a difference of summations yields the expression

$$\liminf_{n\to\infty} \left(\sum_{k=1}^{n} \frac{r_k}{n(1-\gamma)} - \sum_{k=1}^{n} \frac{r_k\gamma^{n+1-k}}{n(1-\gamma)} \right). \tag{7.6}$$

Since the reward sequence generated over a finite environment is made up of only finitely many distinct real numbers, it is necessarily bounded, and we may apply Lemma 7.2. Thus, the subtrahend in the above difference tends to 0 as $n \to \infty$, and so the expression (7.6) is equivalent to

$$\liminf_{n\to\infty} \sum_{k=1}^{n} \frac{r_k}{n(1-\gamma)}.$$

Factoring the constant term in the denominator out of the limit, the remaining limiting expression is exactly the definition of the average payoff, and so we conclude that

$$V^{\gamma}(x) = \frac{V(x)}{1-\gamma}.$$ □

Theorem 7.2 implies the following result on the strategic complexity for the average depreciating payoff.

Corollary 7.5 – Strategic Complexity Of Average Depreciating Optimization

For any average depreciating payoff over any finite MDP, there exists an optimal policy that is stationary and deterministic.

Recall that the value problem, with respect to an MDP M and an objective F, asks to determine, for some state x and bound $b \in \mathbb{Q}$, whether $b \leq \mathrm{Val}_M(\mathsf{F}, x)$. It follows additionally that the average depreciating value problem can be solved either by multiplying each reward occurring in the arena by

$\frac{1}{1-\gamma}$ and solving the average value problem over the arena with scaled rewards, or by solving the average value problem over the given MDP and scaling the result by $\frac{1}{1-\gamma}$.

Corollary 7.6 – Computational Complexity Of Average Depreciating Optimization

Over any finite MDP:

(1) the value problem for average depreciating payoffs is LOGSPACE-reducible to the value problem for average payoff;

(2) values and optimal policies under average depreciating payoffs are computable in polynomial time.

As a direct consequence of Theorem 7.2, there exists a Blackwell optimal policy [FS01] that is optimal for V_λ^γ when λ is sufficiently close to 1, that is also optimal for V^γ.

Corollary 7.7 – Existence of Blackwell Optimal Policies For Depreciating Payoffs

Consider an arbitrary Markov decision process and an arbitrary depreciation factor $\gamma \in [0,1]$. There exists a discount factor $\lambda_0 \in [0,1)$ and a policy π that—for each state x and every $\lambda \in [\lambda_0, 1)$—is optimal with respect to both $\mathsf{D}_\lambda^\gamma$ and A^γ:

$$V_\lambda^\gamma(x) = \mathbb{E}_x^\pi[\mathsf{D}_\lambda^\gamma] \quad \text{and} \quad V^\gamma(x) = \mathbb{E}_x^\pi[\mathsf{A}^\gamma].$$

7.3 Asymptotics

In this section we consider asymptotic relationships discounted depreciating values and average depreciating values that arise from taking limits with respect to the discount and depreciation factors. Fix all of the following arbitrarily: an MDP M, an initial state $x \in X$, a discount factor $\lambda \in [0,1)$, and a depreciation factor $\gamma \in [0,1]$.

There is a natural interpretation of γ as a measure of an agent's memory of past events, and this interpretation is helpful for building intuition. One can think of the standard discounted payoff as a limiting case of the discounted depreciating payoff where $\gamma = 0$ and thus any reward obtained by an agent depreciates to zero as soon as it becomes an asset. In terms of the memory-oriented interpretation, this situation is equivalent to that in which agents' have no recollection ability whatsoever and immediately forget everything but the next immediate reward upon each transition. Conversely, an agent with perfect

memory, *i.e.* where $\gamma = 1$, would end up maximizing a discounted payoff on the sequence of cumulative assets $\left(\sum_{k=1}^{n} r_k\right)_{n\in\mathbb{N}}$ rather than the sequence $(r_n)_{n\in\mathbb{N}}$ of rewards. The following proposition characterizes discounted depreciating values for these extreme cases of minimal and maximal depreciation.

Proposition 7.1

Taking limits $\gamma \to 1$ from below and $\gamma \to 0$ from above,

$$\lim_{\gamma\to 1} V_\lambda^\gamma(x) = \frac{V_\lambda(x)}{1-\lambda} \quad \text{and} \quad \lim_{\gamma\to 0} V_\lambda^\gamma(x) = V_\lambda(x).$$

For the purposes of illustrating Proposition 7.1, Example 7.1 describes a scenario involving a used car dealership.

Example 7.1 – Used Car Dealership

Consider a used car dealership with a business model involving purchasing used cars in locations with favorable regional markets, driving them back to their shop, and selling them for profit in their local market. Suppose that our optimizing agent is an employee of this dealership, tasked with managing inventory acquisition. More specifically, this employee's job is to decide the location from which the next car should be purchased, whenever such a choice arises. Their objective is to maximize the expected cumulative inventory value of the dealership. It is well known [Ack73; Wyk70] that cars tend to continually depreciate in value after being sold as new, and so any reasonable model for the value of all vehicles in the inventory should somehow incorporate an appropriate mechanism to describe this decay. Suppose that a depreciation factor $\gamma \in [0,1]$ captures the rate at which automobiles lose value per unit of time. If the employee assumes a discounted time-horizon for some discount factor $\lambda \in [0,1)$ that quantifies their time preference, then the problem boils down to maximizing $\sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=1}^{n} r_k \gamma^{n-k}$, in which r_k is the initial value of the car purchased at time k.

The situation may be cast as a discounted depreciating optimization problem over a finite Markov decision process, as shown in Figure 7.1[a]. For the sake of simplicity, suppose that there are only two locations x_1 and x_2 from which to choose the next target market. The only point where the employee has more than one possible action is at the dealership x, from where they can choose action a_1 to go to x_1 or a_2 to go to x_2. Realizing that it is unreasonable to plan without expecting unforeseen delays, the employee also introduces two parameters p_1 and p_2, which are success rates for buying a desired vehicle in the locations x_1 and x_2, respectively. Given that the agent is in location x_i, the rate p_i is interpreted as the probability that they find a seller and purchase a vehicle before the end of the day and thus $1 - p_i$ is the probability that they fail to do so. In state t_i, a purchased vehicle is in transit from location x_i to the dealership. Once a vehicle arrives at the dealership, a reward of r_i is obtained, as indicated on the transitions from states t_i to state x.

[a] In the figure, an action is indicated by a lowercase a, a transition probability is indicated by a lowercase p (or an expression involving p), and a reward is indicated by a lowercase r. If an action is omitted from an edge label, then there is only one action available from the edge's source state. If a transition probability is omitted, then the transition is deterministic, *i.e.* occurs with probability 1. If a reward value is omitted, then the reward obtained is 0.

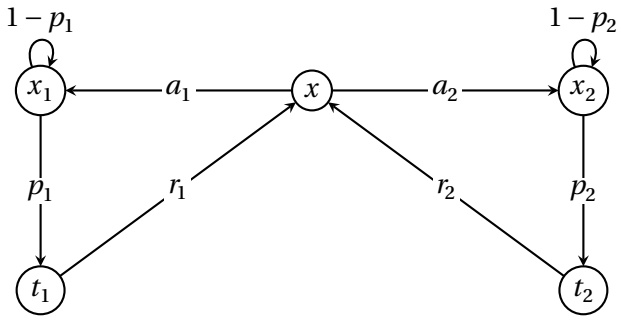
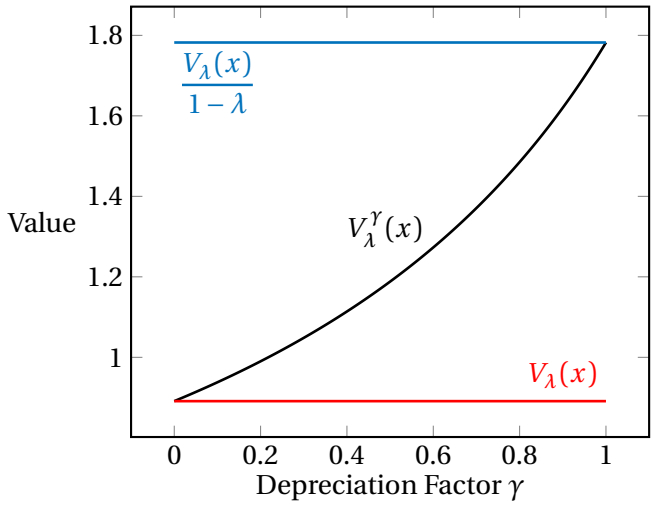


Figure 7.1: A Markov decision process modeling the depreciating optimization problem of the used car dealership from Example 7.1.



Parameters
$\lambda = \frac{1}{2}$
$p_1 = \frac{1}{2}$
$p_2 = \frac{1}{4}$
$r_1 = 5$
$r_2 = 7$

Figure 7.2: A plot of the discounted depreciating value, using the listed parameter values, for the used car dealership from Example 7.1 as the depreciation factor γ varies over the unit interval.

Plugging concrete values in for the parameters of the Example 7.1, we obtain the plot in Figure 7.2, which shows the relationship between the depreciation factor and the optimal discounted payoff of the dealership.

A classic result of Bewley and Kohlberg [BK76][3] establishes that, over a common environment, the product of the λ-discounted value with $(1-\lambda)$ approaches the long-run average value as the discount factor approaches 1 from below:

$$\lim_{\lambda \to 1}(1-\lambda)V_\lambda(x) = V(x).$$

Following suit, we show that this asymptotic relationship between discounted and average values is also

[3] An alternative proof is given by Oliu-Barton [Oli14].

characteristic of their depreciating counterparts.

Theorem 7.3 – Tauberian Theorem For Depreciating Payoffs

As $\lambda \to 1$ from below,

$$\lim_{\lambda \to 1}(1-\lambda)V_\lambda^\gamma(x) = V^\gamma(x).$$

Proof. By Theorem 7.1, we obtain the equation

$$\lim_{\lambda \to 1}(1-\lambda)V_\lambda^\gamma(x) = \lim_{\lambda \to 1}(1-\lambda)\frac{V_\lambda(x)}{1-\lambda\gamma}.$$

Applying Bewley & Kohlberg's theorem to the right-hand side of this equation yields

$$\lim_{\lambda \to 1}(1-\lambda)V_\lambda^\gamma(x) = \frac{V(x)}{1-\gamma}.$$

Finally, it follows from Theorem 7.2 that

$$\lim_{\lambda \to 1}(1-\lambda)V_\lambda^\gamma(x) = V^\gamma(x).$$

□

Chapter 8

Past-Discounted Objectives

This chapter introduces *past-discounted payoffs* in the context of infinite-horizon optimization within the framework of Markov decision processes. Given a finite sequence of rewards and a discount factor $\gamma \in [0,1)$, the past-discounted sum—first considered by [Alu+13]—is defined as

$$\sum_{k=0}^{n} r_k \gamma^{n-k},$$

which contrasts the traditional discounted sum in that the coefficients increase in magnitude with the indices of the sequence.

Despite the symmetry between discounted sums and past-discounted sums, formulating a well-defined notion of past-discounted payoffs on infinite sequences is not quite as simple as it is for discounted sums. Perhaps the most straightforward way to define a past-discounted payoff for an infinite sequence $(r_n)_{n=0}^{\infty}$ of rewards is to take the limit $\lim_{n\to\infty} \sum_{k=0}^{n} r_k \gamma^{n-k}$. Even for bounded sequences, however, this limit may not exist. To see this, consider the bounded infinite sequence $r = 1,2,1,2,\ldots$. The even-indexed terms of the sequence of past-discounted sums of r are equal to

$$2\frac{1-(\gamma^2)^n}{1-\gamma^2} + \gamma\frac{1-(\gamma^2)^n}{1-\gamma^2},$$

while the odd-indexed terms of the sequence of past-discounted sums of r are equal to

$$\frac{1-(\gamma^2)^n}{1-\gamma^2} + 2\gamma\frac{1-(\gamma^2)^n}{1-\gamma^2}.$$

In the limit as $n \to \infty$, the former converges to $\dfrac{2+\gamma}{1-\gamma^2}$ while the latter converges to $\dfrac{1+2\gamma}{1-\gamma^2}$, showing that the sequence as whole does not have a unique limit point. Therefore, the straight forward approach of

defining an objective fails to yield a legitimate past-discounted payoff function. Instead, we consider upper and a lower past-discounted payoffs, defined, respectively, as upper and lower limits of sequences of past-discounted sums:

$$\liminf_{n\to\infty} \sum_{k=0}^{n} r_k \gamma^{n-k} \qquad \text{and} \qquad \limsup_{n\to\infty} \sum_{k=0}^{n} r_k \gamma^{n-k}.$$

To motivate the idea of past-discounting, consider a setting where rewards in a sequence represent the performance of an agent. Under such an interpretation, rewards obtained by the agent at each stage represent the immediate response to their actions from the environment. The past-discounted sum of the finite sequence represents a present-biased cumulative evaluation of the agent in which actions taken more recently are given greater significance. The discount factor γ can be seen as a parameter quantifying the evaluator's time preference. If we suppose the cause of this preference to be that the evaluator has imperfect memory that worsens over time, then γ may represent the rate at which the evaluator forgets or forgives. Generally, past-discounting makes sense conceptually whenever an agent in question can grow and adapt and the evaluator either has a time-decaying memory or another form of recency bias.

Here, we give a precise definition for past-discounted payoffs and study their basic properties.

Definition 8.1 – Past-Discounted Payoffs

Let $\gamma \in [0,1)$ be a discount factor, and let $\mathfrak{r}$ be a reward sequence. The *lower past-discounted payoff* is defined as

$$\mathsf{P}_\gamma^\flat(\mathfrak{r}) = \liminf_{n\to\infty} \sum_{k=1}^{n} r_k \gamma^{n-k},$$

and the *upper past-discounted payoff* is given by

$$\mathsf{P}_\gamma^\sharp(\mathfrak{r}) = \limsup_{n\to\infty} \sum_{k=1}^{n} r_k \gamma^{n-k}.$$

Remark 8.1 – On Notation

At times, it is convenient to refer to a past-discounted payoff without specifying whether it is the upper or lower version. In these situations, we write $\mathsf{P}^{\natural}_{\gamma}$, where the symbol $\natural$ should be understood as representing either of the symbols $\flat$ or $\sharp$. In any given expression, all instances of $\natural$ refer to the same undetermined symbol, *i.e.* an expression may be interpreted such that either (i) all instances of $\natural$ represent $\flat$ or (ii) all instances of $\natural$ represent $\sharp$.
We also write lim opt to denote either lim sup or lim inf, when the distinction is insignificant in a given context.

We begin the investigation by considering the complexity of policies that achieve optimal past-discounted values. In particular, the best understood and most popular payoffs—such as discounted payoffs, mean payoff, and limit payoffs—can be optimized by positional policies.

A general result from the literature concerning strategic complexity of payoffs in MDPs is a theorem of [Gim07] that provides a sufficient condition under which arbitrary payoff functions admit optimal positional policies. In order to state this result precisely, we must recall some definitions from [Gim07].

Definition 8.2 – Prefix-Independence

A payoff F is *prefix-independent* if $\mathsf{F}(\mathfrak{r}\mathfrak{s}) = \mathsf{F}(\mathfrak{s})$ holds for any pair of reward sequences $\mathfrak{r}$ and $\mathfrak{s}$, where $\mathfrak{r}$ is finite and $\mathfrak{s}$ is infinite.

Definition 8.3 – Submixing

A payoff F is *submixing* if the inequality

$$\mathsf{F}(\mathfrak{r}_1\mathfrak{s}_1\mathfrak{r}_2\mathfrak{s}_2\ldots) \leq \max\{\mathsf{F}(\mathfrak{r}_1\mathfrak{r}_2\ldots), \mathsf{F}(\mathfrak{s}_1\mathfrak{s}_2\ldots)\}$$

holds for any pair of infinite sequences of finite real sequences $(\mathfrak{r}_n)_{n=1}^{\infty}$ and $(\mathfrak{s}_n)_{n=1}^{\infty}$.

Theorem 8.1 – Gimbert [Gim07]

Any prefix-independent and submixing payoff admits optimal positional policies over any finite MDP.

During initial attempts to show that past-discounted payoffs admit optimal positional policies, we found that they meet only one of the conditions of the above theorem.

Proposition 8.1 – Past-Discounted Payoffs are Prefix-Independent

If $\mathfrak{r} = r_1, ..., r_m$ is a finite sequence of real numbers and $\mathfrak{s} = (s_k)_{k=1}^{\infty}$ is a bounded sequence of real numbers, then

$$\mathsf{P}_\gamma^\natural(\mathfrak{rs}) = \mathsf{P}_\gamma^\natural(\mathfrak{s}).$$

Proof. Splitting up the summation in Definition 8.1 yields

$$\mathsf{P}_\gamma^\natural(\mathfrak{rs}) = \lim_{n\to\infty} \operatorname{opt}\left(\sum_{k=1}^{m} r_k \gamma^{n-k} + \sum_{k=1}^{n-m} s_k \gamma^{n-m-k}\right).$$

It is a well known fact that lim inf is superadditive and lim sup is subadditive:

$$\liminf_{n\to\infty}(c_n + d_n) \geq \liminf_{n\to\infty} c_n + \liminf_{n\to\infty} d_n \quad \text{and} \quad \limsup_{n\to\infty}(c_n + d_n) \leq \limsup_{n\to\infty} c_n + \limsup_{n\to\infty} d_n,$$

and that these inequalities become equalities if either $(c_n)_{n=1}^{\infty}$ or $(d_n)_{n=1}^{\infty}$ is convergent. Therefore, the equation

$$\mathsf{P}_\gamma^\natural(\mathfrak{rs}) = \lim_{n\to\infty} \operatorname{opt} \sum_{k=1}^{m} r_k \gamma^{n-k} + \lim_{n\to\infty} \operatorname{opt} \sum_{k=1}^{n-m} s_k \gamma^{n-m-k} \tag{8.1}$$

holds if either of the two series being summed is convergent. The left-hand summation converges to zero, since $\sum_{k=1}^{m} \frac{r_k}{\gamma^k}$ is constant with respect to n and $\lim_{n\to\infty} \gamma^n = 0$. This proves that Equation (8.1) indeed holds and, moreover, that

$$\mathsf{P}_\gamma^\natural(\mathfrak{rs}) = \lim_{n\to\infty} \operatorname{opt} \sum_{k=1}^{n-m} s_k \gamma^{n-m-k}.$$

Because m is a fixed constant, it follows that

$$\lim_{n\to\infty} \operatorname{opt} \sum_{k=1}^{n-m} s_k \gamma^{n-m-k} = \lim_{n\to\infty} \operatorname{opt} \sum_{k=1}^{n} s_k \gamma^{n-k},$$

which, in turn, implies $\mathsf{P}_\gamma^\natural(\mathfrak{rs}) = \mathsf{P}_\gamma^\natural(\mathfrak{s})$. □

An immediate corollary of prefix-independence is that all states within a single communicating component have the same past-discounted values.

Corollary 8.1

If x and y are two states that belong to the same communicating component of M and $\pi \in \Pi_M$, then

$$\mathbb{E}_M^{\pi,x}\left(\mathsf{P}_\gamma^\natural\right) = \mathbb{E}_M^{\pi,y}\left(\mathsf{P}_\gamma^\natural\right).$$

In contrast to the property of prefix-independence, past-discounted payoffs are not submixing.

Proposition 8.2 – Past-Discounted Payoffs are Non-Submixing

There exist infinite reward sequences $\mathfrak{s}$, $\mathfrak{q}$, and an interleaving $\mathfrak{r}$ of $\mathfrak{s}$ and $\mathfrak{q}$ such that

$$\mathsf{P}^{\flat}_{\gamma}(\mathfrak{r}) > \max\left\{\mathsf{P}^{\flat}_{\gamma}(\mathfrak{s}), \mathsf{P}^{\flat}_{\gamma}(\mathfrak{q})\right\}.$$

Proof. Consider the pair of periodic sequences $\mathfrak{s} = 2, 1, 200, 100, \ldots$ and $\mathfrak{q} = 200, 100, 2, 1, \ldots$ and an interleaving $\mathfrak{r} = 200, 2, 100, 1, 200, 2, 100, 1, \ldots$. Since (I) the periodically repeating terms defining $\mathfrak{s}$ and $\mathfrak{q}$ are rotations of each other and (II) past-discounted payoffs are prefix-independent, the sequence of past-discounted sums of $\mathfrak{s}$ and $\mathfrak{q}$ have the same set of limit points:

$$\left\{\frac{2\gamma^3+\gamma^2+200\gamma+100}{1-\gamma^4}, \frac{100\gamma^3+2\gamma^2+\gamma+200}{1-\gamma^4}, \frac{200\gamma^3+100\gamma^2+2\gamma+1}{1-\gamma^4}, \frac{\gamma^3+200\gamma^2+100\gamma+2}{1-\gamma^4}\right\}.$$

Thus, the upper (resp. lower) past-discounted values of $\mathfrak{s}$ and $\mathfrak{q}$ are equivalent and equal to the maximum (resp. minimum) of the quantities in the above set. The set of limit points of past-discounted sums of $\mathfrak{r}$ is

$$\left\{\frac{200\gamma^3+2\gamma^2+100\gamma+1}{1-\gamma^4}, \frac{\gamma^3+200\gamma^2+2\gamma+100}{1-\gamma^4}, \frac{100\gamma^3+\gamma^2+200\gamma+2}{1-\gamma^4}, \frac{2\gamma^3+100\gamma^2+\gamma+200}{1-\gamma^4}\right\}.$$

Setting $\gamma = 1/10$, we can evaluate these payoffs on the given reward sequences to

$$\mathsf{P}^{\sharp}_{\gamma}(\mathfrak{s}) = \mathsf{P}^{\sharp}_{\gamma}(\mathfrak{q}) \approx 200.24 \qquad \text{and} \qquad \mathsf{P}^{\sharp}_{\gamma}(\mathfrak{r}) \approx 201.1221,$$

which shows that

$$\mathsf{P}^{\sharp}_{\gamma}(\mathfrak{r}) > \max\left\{\mathsf{P}^{\sharp}_{\gamma}(\mathfrak{s}), \mathsf{P}^{\sharp}_{\gamma}(\mathfrak{q})\right\}.$$

Likewise, the lower past-discounted values of these sequences are

$$\mathsf{P}^{\flat}_{\gamma}(\mathfrak{s}) = \mathsf{P}^{\flat}_{\gamma}(\mathfrak{q}) \approx 2.4002 \qquad \text{and} \qquad \mathsf{P}^{\flat}_{\gamma}(\mathfrak{r}) \approx 11.2211,$$

which implies the inequality

$$\mathsf{P}^{\flat}_{\gamma}(\mathfrak{r}) > \max\left\{\mathsf{P}^{\flat}_{\gamma}(\mathfrak{s}), \mathsf{P}^{\flat}_{\gamma}(\mathfrak{q})\right\}.$$

Hence, we conclude that neither upper nor lower past-discounted payoffs are submixing. □

8.1 Past-Discounting In Deterministic Environments

Proposition 8.2 suggests that past-discounted payoffs may not admit stationary deterministic optimal policies in general. Indeed, the following result establishes that finite-memory policies are not sufficient for achieving optimality with respect to the upper past-discounted payoff, even in the restricted case of deterministic MDPs.

Recall that an MDP $M = \langle X, A, P, R\rangle$ is deterministic if $P(y \mid x, a) \in \{0, 1\}$ for all $x, y \in X$ and $a \in A$. In deterministic environments, optima are invariably achieved by deterministic policies, and so we restrict our attention in the following proof to deterministic policies. In such settings, a policy π induces a unique trajectory $\mathfrak{t}$ for each state x in the process. For convenience, we write $\pi(\mathfrak{p}) = a$ as shorthand when $\pi(a \mid \mathfrak{p}) = 1$.

The value of a policy π, with respect to a payoff F, is not really an expectation in this context, so we write

$$\mathrm{Val}_M^\pi(\mathsf{F}, x) = \mathsf{F}(\mathfrak{t}).$$

Hence, we rephrase the definition of the value of F from a state x according to the equation

$$\mathrm{Val}_M(\mathsf{F}, x) = \sup_{\pi \in \Pi_M^D} \mathrm{Val}_M^\pi(\mathsf{F}, x).$$

Theorem 8.2 – History Dependence Of Optimal Policies

Optimal policies for upper past-discounted payoffs require unbounded memory in general.

Proof. Consider the deterministic MDP depicted in Figure 8.1. There are two stationary policies:

- π which selects action a from state y and
- ρ which selects action b from state y.

Omitting finite prefixes, as permitted by Proposition 8.1, the trajectory generated by π is $\mathfrak{t} = (\mathfrak{t}_n)_{n=1}^\infty = (y)_{n=1}^\infty$, with corresponding reward sequence $R(\mathfrak{t}) = R_\mathfrak{t} = \ldots, 1, 1, 1, 1, \ldots$, and the trajectory generated by ρ is $\mathfrak{u} = \ldots, x, y, x, y, \ldots$, having $R(\mathfrak{u}) = R_\mathfrak{u} = \ldots, 0, 3, 0, 3, \ldots$ as its reward sequence.

Since $R(\mathrm{t})$ is a constant sequence, the sequence of past-discounted sums converges such that

$$\mathsf{P}^{\sharp}_{\gamma}(r_t) = \lim_{n\to\infty} \sum_{k=1}^{n} \gamma^{n-k} = \frac{1}{1-\gamma}.$$

The reward sequence $R_{\mathrm{u}} = (r_n)_{n=1}^{\infty}$ is not constant, but it may be decomposed into two constant sequences by grouping terms based on the parity of their indices. Then, the past discounted sums of $R(\mathrm{u})$ may be written as

$$\sum_{k=1}^{2n} r_k \gamma^{2n-k} = \sum_{k=1}^{n} r_{2k-1}\gamma^{2(n-k)+1} + \sum_{k=1}^{n} r_{2k}\gamma^{2(n-k)}.$$

Interchanging indices of the reward terms and the exponents of the discount factor, this is equivalent to the expression

$$\sum_{k=1}^{n} r_{2(n-k)+1}\gamma^{2k-1} + \sum_{k=1}^{n} r_{2(n-k+1)}\gamma^{2(k-1)}.$$

One of the summations in the above expression will have all coefficients be 0, while the other will have all coefficients as 3. We can safely discard the 0 summation, which leaves, depending on whether $r_{2n} = 3$ or $r_{2n-1} = 3$, one of

$$\sum_{k=1}^{n} 3\gamma^{2k-1} \quad \text{or} \quad \sum_{k=1}^{n} 3\gamma^{2(k-1)}.$$

Therefore, the sequence of past-discounted sums of $R(\mathrm{u})$ has exactly two convergent subsequences, and so the upper past-discounted payoff on x is equal to the greatest of these limit points:

$$\mathsf{P}^{\sharp}_{\gamma}(R_{\mathrm{u}}) = \max\left\{\frac{3}{1-\gamma^2}, \frac{3\gamma}{1-\gamma^2}\right\}.$$

Letting $\gamma = 1/3$, the upper past-discounted values of the positional policies π and ρ evaluate, respectively, to

$$\mathrm{Val}^{\pi}_{M}\left(\mathsf{P}^{\sharp}_{\gamma}, x\right) = \mathrm{Val}^{\pi}_{M}\left(\mathsf{P}^{\sharp}_{\gamma}, y\right) = 1\,1/2 \qquad \text{and} \qquad \mathrm{Val}^{\rho}_{M}\left(\mathsf{P}^{\sharp}_{\gamma}, x\right) = \mathrm{Val}^{\rho}_{M}\left(\mathsf{P}^{\sharp}_{\gamma}, y\right) = 3\,3/8.$$

To complete the proof, we identify a family of finite-memory policies having values that grow in proportion to the constant bounding their memory. Consider the finite-memory policy π_m, for $m\in\mathbb{N}$, that generates the trajectory

$$\mathrm{t}_m = \left(\mathrm{t}_{m,n}\right)_{n=1}^{\infty} = \ldots, x, \underbrace{y, \ldots, y}_{m \text{ times}}, x, \underbrace{y, \ldots, y}_{m \text{ times}}, x \ldots$$

with reward sequence

$$\mathfrak{s} = (s_n)_{n=1}^{\infty} = \ldots, 0, \underbrace{1, \ldots, 1}_{m-1 \text{ times}}, 3, 0, \underbrace{1, \ldots, 1}_{m-1 \text{ times}}, 3, \ldots.$$

The n-step past-discounted sum of $\mathfrak{s}$ can be split up as

$$\sum_{k=n-m-1}^{n} s_k \gamma^{n-k} + \sum_{k=1}^{n-m-2} s_k \gamma^{n-k},$$

and interchanging indices in the left-hand summation yields

$$\sum_{k=1}^{m+1} s_{n-k} \gamma^{k-1} + \gamma^{m+2} \sum_{k=1}^{n-m-2} s_k \gamma^{n-m-2-k}.$$

Supposing that $\mathfrak{t}_{m,n+1} = x$, this is equivalent to

$$3 + \gamma \sum_{k=1}^{m-1} \gamma^{k-1} + \gamma^{m+2} \sum_{k=1}^{n-m-2} s_k \gamma^{n-m-2-k},$$

which, in turn, may be rewritten as

$$3 + \frac{\gamma(1-\gamma^{m-1})}{1-\gamma} + \gamma^{m+2} \sum_{k=1}^{n-m-2} s_k \gamma^{n-m-2-k}. \tag{8.2}$$

The limit of this expression as $n \to \infty$ is the greatest amongst the $m+1$ limit points of the sequence of past-discounted sum of s. Letting ϵ represent the summation on the right, (8.2) may be simplified to

$$3 + \frac{\gamma(1-\gamma^{m-1})}{1-\gamma} + \gamma^{m+2}\epsilon.$$

Hence, when $\gamma = \frac{1}{3}$ and $m = 2$, we get that

$$\mathrm{Val}_M^{\pi_2}\left(\mathrm{P}_\gamma^\sharp, x\right) = \mathrm{Val}_M^{\pi_2}\left(\mathrm{P}_\gamma^\sharp, y\right) = 4^1/_3 + \frac{\epsilon}{81}, \tag{8.3}$$

which is already greater than either of the upper past-discounted values for the stationary policies π and ρ. As m increases, it is clear that the middle summand in (8.3) increases while the right-most summand decreases. Thus, we conclude that, for every $m \in \mathbb{N}$,

$$\mathrm{Val}_M^{\pi_m}\left(\mathrm{P}_\gamma^\sharp, x\right) < \mathrm{Val}_M^{\pi_{m+1}}\left(\mathrm{P}_\gamma^\sharp, x\right). \qquad \square$$

Remark 8.2

The careful reader may notice that the proof of Theorem 8.2 fails for the lower past-discounted payoff. We do not know whether analogous counterexamples refuting the optimality of positional policies exist in this case.

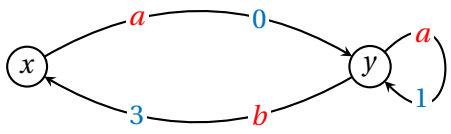


Figure 8.1: A deterministic MDP for which optimal past-discounted policies require unbounded memory. Actions are displayed in red, rewards are displayed in blue.

The rest of this section continues the investigation of upper past-discounted payoffs over deterministic MDPs. In this setting, we analyze the computational complexity of computing past-discounted values and construct an algorithm for synthesis of optimal non-stationary policies. We begin with the following observation.

A deterministic MDP $\langle X, A, P, R\rangle$ is essentially a directed graph $G = \langle V, E, L, R\rangle$in which,

- a vertex in V is a state in X;
- an edge in E is an ordered pair of states $\langle x, y\rangle$ for which there is an action a such that $P(y \mid x, a) = 1$;
- each edge $\langle x, y\rangle$ is labeled by the action $L(x, y) = a$ for which $P(y \mid x, a) = 1$;
- each edge $\langle x, y\rangle$ is weighted by the reward $R(x, y)$.

With this correspondence in mind, we refer to deterministic MDPs as digraphs or simply as graphs in the following. The terms "state" and "vertex" will also be used interchangeably.

Let us now fix some terminology, borrowed from graph theory. A *cycle* is a path that begins and ends with a common states but has no other states occurring twice. If there exists a path beginning at x and ending at y, then y is said to be *reachable* from x. Two states x and y are *connected* if each is reachable from the other. Connectivity is an equivalence relation over the set of states and therefore determines a partition based on the associated equivalence classes. A *subgraph* of a given graph is obtained by removing a subset of vertices and any adjacent edges. For any subset of states $Y \subseteq X$, the subgraph of $G = \langle X, A, P, R\rangle$ induced by Y, written G_Y, is the subgraph obtained by removing all vertices in $X \setminus Y$ and any adjacent edges. A *strongly connected component* (SCC) of a graph is a subgraph induced by an equivalence class of the connectivity relation.

Strongly Connected Digraphs

A strongly connected digraph is a graph whose vertices comprise a single strongly connected component. Here, we establish results about strongly connected digraphs that are subsequently lifted to general graphs. We make use of a transpose operation that reverses the edges of a strongly connected digraph.

Definition 8.4

The transpose of a strongly connected digraph $G = \langle V, E, L, R\rangle$ is another strongly connected digraph $G^\intercal = \langle V, E^\intercal, L^\intercal, R^\intercal\rangle$, where

- $E^\intercal = \{\langle x, y\rangle \in V \times V : \langle x, y\rangle \in E\}$;
- $L^\intercal(x, y) = L(y, x)$;
- $R^\intercal(x, y) = R(y, x)$.

The next result equates the upper past-discounted value of a vertex in a strongly connected graph to the maximal discounted value of any vertex in its transpose.

Theorem 8.3

For any vertex x in a strongly connected digraph G,

$$\mathrm{Val}_G\left(\mathsf{P}^\sharp_\gamma, x\right) = \max_{y\in V} \mathrm{Val}_{G^\intercal}\left(\mathsf{D}_\gamma, y\right).$$

Proof. The proof proceeds in two parts: Lemma 8.1 upper bounds the upper past-discounted value by $\max_{y\in V} \mathrm{Val}_{G^\intercal}\left(\mathsf{D}_\gamma, y\right)$ and then Lemma 8.2 establishes the existence a policy $\pi \in \Pi^D_G$ such that $\mathrm{Val}^\pi_G\left(\mathsf{P}^\sharp_\gamma, x\right)$ achieves this upper bound. □

Lemma 8.1

For any vertex x in a strongly connected digraph G,

$$\mathrm{Val}_G\left(\mathsf{P}^\sharp_\gamma, x\right) \le \max_{y\in V} \mathrm{Val}_{G^\intercal}\left(\mathsf{D}_\gamma, y\right).$$

Proof. Because G is strongly connected, so is $G^\intercal$, and, for any path in G, the reversed path is in $G^\intercal$. This implies that

$$\sum_{k=0}^{n-1} R(y_k, y_{k+1})\gamma^{n-k} = \sum_{k=0}^{n-1} R^\intercal(y_{n-k}, y_{n-k-1})\gamma^k.$$

Therefore, the sequence of past-discounted sums of any trajectory in over G is identical to an infinite sequence of discounted sums of paths in $G^\intercal$. Consequently, we can bound the upper past-discounted value from above by the maximal discounted value of any vertex in $G^\intercal$ as

$$\mathrm{Val}_G\left(\mathsf{P}_\gamma^\sharp, x\right) \leq \max_{y \in V} \mathrm{Val}_{G^\intercal}\left(\mathsf{D}_\gamma, y\right). \qquad \square$$

Lemma 8.2

Over any strongly connected digraph G, there exists a policy π such that the equation

$$\mathrm{Val}_G^\pi\left(\mathsf{P}_\gamma^\sharp, x\right) = \max_{y \in V} \mathrm{Val}_{G^\intercal}\left(\mathsf{D}_\gamma, y\right)$$

holds for every state x

Proof. Suppose that $\mathfrak{t} = (x_n)_{n=1}^\infty$ is the trajectory generated by a policy ρ that is optimal for D_γ over $G^\intercal$ from the vertex $\arg\max_{y \in V} \mathrm{Val}_{G^\intercal}\left(\mathsf{D}_\gamma, y\right)$. Let π be a policy generating trajectory $\mathfrak{u}$ for which there exists a map $f : \mathbb{N} \to \mathbb{N}$ such that $\mathfrak{t}_{[f(n),0]} \in \mathrm{Inf}(\mathfrak{u})$, for all $n \in \mathbb{N}$. Let $g : \mathbb{N} \to \mathbb{N}$ be the map defined by the equation

$$\mathfrak{t}_{[f(n),0]} = \mathfrak{u}_{[g(n),g(n)+f(n)]}.$$

By definition, we have the equation

$$\mathrm{Val}_G^\pi\left(\mathsf{P}_\gamma^\sharp, x\right) = \limsup_{n \to \infty} \sum_{k=1}^{n} R(x_{k-1}, x_k)\gamma^{n-k}.$$

Now, consider the finite summation

$$\sum_{k=1}^{f(n)+g(n)} R(x_{k-1}, x_k)\gamma^{f(n)+g(n)-k}.$$

Letting $h(n) = f(n) + g(n)$, this sum may be rewritten as

$$\sum_{k=1}^{g(n)} R(x_{k-1}, x_k)\gamma^{h(n)-k} + \sum_{k=g(n)+1}^{h(n)} R(x_{k-1}, x_k)\gamma^{h(n)-k}. \tag{8.4}$$

By Definition 8.4, the right-hand summation is equivalent to

$$\sum_{k=g(n)+1}^{h(n)} R^\intercal(x_k, x_{k-1})\gamma^{h(n)-k}.$$

Interchanging the indices of the two terms forming the product within this sum yields

$$\sum_{k=g(n)+1}^{h(n)} R^\intercal(x_{h(n)-k}, x_{h(n)-k-1})\gamma^k.$$

Applying the definition of g, to this sum, in turn, we obtain the equivalent expression

$$\sum_{k=1}^{f(n)} R^\intercal(y_{k-1}, y_k)\gamma^k.$$

Substituting this back into (8.4) in place of the right-hand summation and factoring $\gamma^{f(n)}$ from the left-hand summation yields

$$\gamma^{f(n)} \sum_{k=1}^{g(n)} R(x_{k-1}, x_k)\gamma^{g(n)-k} + \sum_{k=1}^{f(n)} R^\intercal(y_{k-1}, y_k)\gamma^k. \tag{8.5}$$

In the limit as $n \to \infty$, the left-hand sum vanishes since $\gamma^{f(n)} \to 0$. The right-hand sum approaches $\mathrm{Val}_{G^\intercal}\left(\mathsf{D}_\gamma, y^\sharp\right)$ as $n \to \infty$. Therefore, the limit as $n \to \infty$ of (8.5) is $\mathrm{Val}_{G^\intercal}\left(\mathsf{D}_\gamma, y^\sharp\right)$. This shows that $\mathrm{Val}_{G^\intercal}\left(\mathsf{D}_\gamma, y^\sharp\right)$ is a subsequential limit of the sequence of past discounted sums with respect to $\mathfrak{t}$. In combination with Lemma 8.1, this implies that

$$\mathrm{Val}_G\left(\mathsf{P}_\gamma^\sharp, x\right) \leq \mathrm{Val}_G^\pi\left(\mathsf{P}_\gamma^\sharp, x\right) = \mathrm{Val}_{G^\intercal}\left(\mathsf{D}_\gamma, y^\sharp\right). \qquad \square$$

As a corollary of Lemma 8.2, we obtain a sufficient condition for upper past-discounted optimality.

Corollary 8.2

Let G be a strongly connected digraph and let $x = \arg\max_{x \in V} \mathrm{Val}_{G^\intercal}\left(\mathsf{D}_\gamma, y\right)$. Suppose that $\mathfrak{t}$ is the trajectory generated by a policy ρ for which $\mathrm{Val}_{G^\intercal}^\rho\left(\mathsf{D}_\gamma, x\right) = \mathrm{Val}_{G^\intercal}\left(\mathsf{D}_\gamma, x\right)$, and $\mathfrak{u}$ is the trajectory generated by a policy π from vertex x. If there exists $f : \mathbb{N} \to \mathbb{N}$ such that $\mathfrak{t}_{[f(n),0]} \in \mathrm{Inf}(\mathfrak{u})$, for all $n \in \mathbb{N}$, then

$$\mathrm{Val}_G^\pi\left(\mathsf{P}_\gamma^\sharp, x\right) = \mathrm{Val}_G\left(\mathsf{P}_\gamma^\sharp, x\right).$$

Since discounted values over finite MDPs are computable in polynomial time [Put94; FV96; FS01], we obtain the following corollary from Theorem 8.3,

Corollary 8.3

Over strongly connected digraphs, upper past-discounted values are computable in polynomial time.

We now turn to the problem of policy synthesis. Algorithm 6 gives a synthesis procedure that takes as input a strongly connected digraph G and returns a policy $\pi_* \in \Pi_G^D$ that is optimal for $\mathsf{P}_\gamma^\sharp$ from any initial vertex. The first portion of the algorithm computes an optimal policy ρ for D_γ over G^T from a vertex $y^\sharp$ having of maximal value and optimal reachability policies ρ_x for each vertex x in G. The policy π_* will use these policies as subroutines, and will also make use of two counters c and d that are maintained in external (to Algorithm 6, that is) memory. Let $\mathfrak{t}$ be the trajectory in G^T generated by ρ from $y^\sharp$, and let $\mathfrak{u}$ be the trajectory in G that gets generated by π_* from some arbitrary initial state. The counter c is only ever incremented and keeps track of the length of the reversed prefix of the trajectory $\mathfrak{t}$ that should be generated next as a subsequence of the trajectory $\mathfrak{u}$. The counter d is used to determine when the path (*i.e.* the prefix of $\mathfrak{u}$) generated so far ends with the reversed prefix $\mathfrak{t}_{[c,0]}$.

Algorithm 6: Optimal policy synthesis for upper past-discounted payoff over a strongly connected digraph.

Input: strongly connected digraph G.

Compute the optimal policy ρ_v over G for reachability with respect to each vertex $x \in V$;
Compute the optimal policy φ for D_γ over G^T from vertex $x^\sharp = \arg\max_{x \in V} \mathrm{Val}_{G^\mathsf{T}}\left(\mathsf{D}_\gamma, x\right)$;
Initialize counters c and d, respectively, to 1 and 0;
Function $\pi_*(x)$**:**
    $x := \varphi^c\left(x^\sharp\right)$;
    if $d = 0$ *and* $x \neq x$ **then**
        return $\rho_u(x)$
    if $c = d$ **then**
        $c := c + 1$;
        $d := 0$;
        $x := \varphi^c\left(x^\sharp\right)$;
        return $\rho_u(x)$;
    $d := d + 1$;
    return $\varphi^{c-d}\left(x^\sharp\right)$;
return π_*;

Next, we establish correctness of Algorithm 6.

Theorem 8.4

Given a strongly connected digraph G, the policy π_* returned by Algorithm 6 is optimal for $\mathsf{P}_\gamma^\sharp$ from any vertex x:

$$\mathrm{Val}_G^{\pi_*}\left(\mathsf{P}_\gamma^\sharp, x\right) = \mathrm{Val}_G\left(\mathsf{P}_\gamma^\sharp, x\right).$$

Proof. Let c_n, d_n be the values of the counters at time n. Notice that $c_n \leq c_{n+1}$ holds for all $n \in \mathbb{N}$, because c is only ever incremented. Furthermore, c is only incremented when $c = d$. In this case, d is also reset to 0; otherwise, after some finite time at value 0, d is incremented. Since c is initialized to 1 and d to 0, this proves inductively that $d_n \leq c_n$, for all $n \in \mathbb{N}$ and that $\{c_n : n \in \mathbb{N}\} = \mathbb{N}$. Therefore, without loss of generality, we can define a function $g : \mathbb{N} \to \mathbb{N}$ in terms of the counter c as $g(c) = \arg_{n \in \mathbb{N}}(c_n = c = d_n)$. After c is incremented and d is set to 0, it is kept at 0 until the last vertex of $\mathfrak{t}_{[0,c]}$ is reached. Then it is incremented on each invocation of π_* until $c = d$, and each of these calls to π_* returns $\pi_*(x) = \rho^{c-d}(x)$. Hence, the last c-length suffix of $\mathfrak{u}$ is precisely

$$\rho^c(x^\sharp), \rho^{c-1}(x^\sharp), \ldots, \rho(x^\sharp), x^\sharp = \mathfrak{t}_{[c,0]} = \mathfrak{u}_{[g(c)-c,g(c)]}$$

at time $g(c)$. This proves that each finite prefix of $\mathfrak{t}$ occurs reversed infinitely often in $\mathfrak{u}$. In other words, taking $f : \mathbb{N} \to \mathbb{N}$ as the identity function $f(n) = n$, it follows that $\mathfrak{t}_{[f(n),0]} \in \mathrm{Inf}(\mathfrak{u})$ for all $n \in \mathbb{N}$. Thus, Coroallary 8.2 may be applied to conclude that π_* is optimal for $\mathsf{P}^\sharp_\gamma$ in G. □

Since the size of the function π_* is independent of the size of the input graph G, the computational complexity of optimal policy synthesis is derived from the computational complexity of synthesizing the policies ρ and $\{\rho_x : x \in V\}$ used to construct π_*. Theorem 8.4 in conjunction with the well-known facts that optimal policies for discounted payoffs [FV96; FS01; Put94] and reachability objectives (via shortest-path algorithms [Cor+09b]) are computable in polynomial time, imply the following corollary.

Corollary 8.4

Over strongly connected digraphs, optimal policies for upper past-discounted payoffs can be synthesized in polynomial time.

General Digraphs

We now use the results developed for upper past-discounted optimization in strongly connected digraphs to bound the computational complexity of computing past-discounted values and synthesizing optimal policies in arbitrary digraphs.

Theorem 8.5

For each vertex x of a digraph G, the value $\mathrm{Val}_G\left(\mathsf{P}^\sharp_\gamma, x\right)$ is computable in polynomial time.

Proof. When the environmental digraph is strongly connected, computing $\mathrm{Val}_G\left(\mathsf{P}^\sharp_\gamma, x\right)$, for any $x \in V$ reduced to computing $\max_{x\in V} \mathrm{Val}_{G^\intercal}\left(\mathsf{D}_\gamma, x\right)$, as established by Theorem 8.3. In the case of general digraphs, there may be many strongly connected components that are reachable from the initial vertex x, each of which has a well-defined transpose. To handle this, we can compute an SCC decomposition $\{C_1, ..., C_n\}$ of G and then compute the maximal discounted value

$$D_k = \max_{x\in C_k} \mathrm{Val}_{G_{C_k^\intercal}}\left(\mathsf{D}_\gamma, x\right)$$

on the transpose of each resulting component. The value $\mathrm{Val}_G\left(\mathsf{P}^\sharp_\gamma, x\right)$ is then equal to the maximal such D_k amongst all the strongly connected components that are reachable in G from x. As shown by [Tar72] and [Sha81], an SCC decomposition can be computed in linear time, the discounted values on the transposes of the resulting strongly connected components can be computed in polynomial time as previously noted, and reachability can be done in polynomial time. Hence, $\mathrm{Val}_G\left(\mathsf{P}^\sharp_\gamma, x\right)$ is computable in polynomial time. □

From the proof of Theorem 8.5, it becomes clear that an optimal policy for $\mathsf{P}^\sharp_\gamma$ in a general digraph has two phases:

(1) choosing from the reachable strongly connected components of the environment the one having the greatest discounted value on its transpose;

(2) carrying out the policy returned by Algorithm 6 within the confines of this strongly connected component.

Since such a policy can be attained similarly to how the value was computed in the above proof, the computational complexity of synthesizing optimal policies past-discounted payoffs is inherited from Theorem 8.5.

Corollary 8.5

Over general digraphs, optimal policies for upper past-discounted payoffs can be synthesized in polynomial time.

8.2 Past-Discounting In Ergodic Environments

Despite the fact that stationary policies are insufficient for optimizing past-discounted payoffs in general, it may still be of interest to obtain the best policy from the restricted class of stationary policies and compute the corresponding (likely suboptimal) value. In this section, we demonstrate how this can be done for the subclass of ergodic MDPs.

Whenever the limit

$$\lim_{n\to\infty} \sum_{\mathfrak{p}\in \mathrm{Path}_n(M)} P_\pi(x \mid \mathfrak{p})$$

exists for every state x of a Markov chain $M = (X, P)$, the resulting distribution over X is called the *limiting distribution* of M. Limiting distributions do not always exist, however, so we focus on a subclass of environments for which they do. An MDP M is called *ergodic* if M_π is an ergodic Markov chain for any stationary policy $\pi \in \Pi^S_M$. Ergodicity is a sufficient condition for the existence of a limiting distribution in a Markov chain; the induced chain M_π has a limiting distribution for any policy $\pi \in \Pi^S_M$ in an ergodic MDP. The following theorem of [Aks+13] is useful.

Theorem 8.6 – Akshay, Bertrand, Haddad, and Hélouët [Aks+13]

If M is an ergodic MDP and, for $\pi \in \Pi_M$, the limiting distribution of M_π exists, then there is a stationary policy ρ such that M_ρ has the same limiting distribution.

Thus, we may restrict our attention to stationary policies when optimizing over ergodic MDPs.

For the remainder of the section, fix an ergodic MDP $M = \langle X, A, P, R\rangle$ and a stationary policy π over M. Let

$$\Delta(x) = \lim_{n\to\infty} \sum_{y\in X} P^n_\pi(x \mid y)$$

be the associated limiting distribution of M_π. Using π, it is possible to define a expected reward for arbitrary time n as

$$\mathbb{E}_\pi(r) = \sum_{x\in X}\sum_{y\in X} \Delta(x)\Delta(y)R(x,y),$$

and this allows us to bring the expectation operator inside the limit in the definitions of past-discounted payoffs:

$$\mathbb{E}_M^{\pi,x}\left(\mathsf{P}_\gamma^\natural\right) = \lim_{n\to\infty} \operatorname{opt} \sum_{k=1}^{n} \mathbb{E}_\pi(r)\gamma^{n-k}.$$

Since $\mathbb{E}_\pi(r)$ is constant, the limit on the right-hand side of the above equation exists such that

$$\mathbb{E}_M^{\pi,x}\left(\mathsf{P}_\gamma^\natural\right) = \frac{\mathbb{E}_\pi(r)}{1-\gamma}.$$

As a result, standard techniques such as linear programming can be employed to compute optimal (amongst the class of stationary policies) values and their associated stationary policies and limiting distributions.

Theorem 8.7

Optimization amongst stationary policies for past-discounted payoffs in finite ergodic MDPs can be done in polynomial time.

8.3 Approximate Past-Discounting In General Environments

Inspired by the work of [BGR19; Cha+15; Cha+13], we propose *window past-discounted payoffs* as a means of approximately solving past-discounted optimization problems in general MDPs.

Definition 8.5

Given a discount factor $\gamma \in [0,1)$ and a window length $\ell \in \mathbb{N}$, the window past-discounted payoffs are defined on a reward sequence $r = (r_n)_{n=0}^{\infty}$ as

$$\mathsf{W}_\ell \mathsf{P}_\gamma^\flat(r) = \liminf_{n\to\infty} \sum_{k=\max(1,n-\ell+1)}^{n} r_k \gamma^{n-k},$$
$$\mathsf{W}_\ell \mathsf{P}_\gamma^\sharp(r) = \limsup_{n\to\infty} \sum_{k=\max(1,n-\ell+1)}^{n} r_k \gamma^{n-k}.$$

The following proposition is immediate from Definition 8.5.

Proposition 8.3

For $\gamma \in [0,1)$ and reward sequence r,

$$\lim_{\ell\to\infty} \mathsf{W}_\ell \mathsf{P}_\gamma^\natural(r) = \mathsf{P}_\gamma^\natural(r) \quad \text{and} \quad \mathsf{W}_0 \mathsf{P}_\gamma^\natural(r) = \mathsf{L}^\natural(r).$$

The next result reduces the computation of window past-discounted values to the computation of limit payoff values over derived MDPs.

Theorem 8.8

Let γ be a discount factor and ℓ be a window length. For any finite MDP $M = \langle X, A, P, R\rangle$, there exists another finite MDP M^ℓ with such that for every state x in M, there exists a state y in M^ℓ such that

$$\mathrm{Val}_M\left(\mathsf{W}_\ell \mathsf{P}^\natural_\gamma, x\right) = \mathrm{Val}_{M^\ell}\left(\mathsf{L}^\natural, y\right).$$

Proof. Let $\mathcal{A} = \langle \Sigma, Q, q_0, \delta\rangle$ be a DFA[1] with

- alphabet $\Sigma = X$,
- state set $Q = \bigcup_{k=1}^{\ell} \mathrm{Path}_k(M) \cup \{q_0\}$, and
- transition function

$$\delta(q,x) = \begin{cases} x & \text{if } q = q_0, \\ qx & \text{if } |q| < \ell, \\ q_{[1,\ell]}x & \text{if } |q| = \ell. \end{cases}$$

Now, let $M^\ell = \left\langle X^\ell, A, P^\ell, R^\ell \right\rangle$ be the MDP constructed as the product of M and $\mathcal{A}$, where

- $X^\ell = \left\{ \langle x, q\rangle \in X \times Q : 0 < \sum_{a\in A} P(x \mid \mathsf{Last}(q), a) \right\}$;
- $P^\ell\left(\left\langle y, q'\right\rangle \mid \langle x, q\rangle, a\right) = \begin{cases} P(y \mid x, a) & \text{if } \delta(q,x) = q' \\ 0 & \text{otherwise,} \end{cases}$
- $R^\ell\left(\langle x, q\rangle, \left\langle y, q'\right\rangle\right) = R(x,y) + \sum_{k=1}^{m} R(x_{k-1}, x_k)\gamma^{m-k}$, where $q' = x_0 \ldots x_m$ and $x_m = x$.

We proceed by showing that for any state x and policy π over M, there exists a policy ρ over M^ℓ such that $\mathbb{E}_M^{\pi,x}\left(\mathsf{W}_\ell \mathsf{P}^\natural_\gamma\right) = \mathbb{E}_{M^\ell}^{\rho,\langle x, q_0\rangle}\left(\mathsf{L}^\natural\right)$. Consider an arbitrary trajectory $\mathfrak{u} = (x_n)_{n=0}^\infty \in \mathrm{Traj}(M_\pi)$. At time n, the probability of the prefix $\mathfrak{u}_{[0,n]}$ is

$$\mathbb{P}^n_{M_\pi}(\mathfrak{u}_{[0,n]}) = \prod_{k=1}^{n} \sum_{a\in A} P(x_k \mid x_{k-1}, a)\pi(a \mid \mathfrak{u}_{[0,k-1]}).$$

[1] We omit any specification of accepting states, because they are not relevant to the proof.

By the construction above, there is a corresponding trajectory $\mathfrak{t} \in \mathrm{Traj}\left(M^\ell\right)$ where $\mathfrak{t}_{[0,n]}$ is

$$\langle x_0, q_0\rangle, \ldots, \left\langle x^\ell, \mathfrak{u}_{[0,\ell-1]}\right\rangle, \ldots, \langle x_n, \mathfrak{u}_{[n-\ell,n-1]}\rangle.$$

under a policy $\rho \in \Pi_{M^\ell}$, this prefix has probability

$$\mathbb{P}^n_{M^\ell_\rho}(\mathfrak{t}_{[0,n]}) = \prod_{k=1}^{n} \sum_{a\in A} P^\ell(\langle x_k, q_k\rangle \mid \langle x_{k-1}, q_{k-1}\rangle, a)\rho(a \mid \mathfrak{t}_{[0,k-1]}).$$

By definition of P^ℓ, we have that

$$P^\ell(\langle x_k, q_k\rangle \mid \langle x_{k-1}, q_{k-1}\rangle, a) = P(x_k \mid x_{k-1}, a)$$

holds for every k. This implies, for all n, that the equation $\mathbb{P}^n_{M_\pi}(\mathfrak{u}_{[0,n]}) = \mathbb{P}^n_{M^\ell_\rho}(\mathfrak{t}_{[0,n]})$ holds precisely when does

$$\pi(a \mid \mathfrak{u}_{[0,k]}) = \rho(a \mid \mathfrak{t}_{[0,k]}). \tag{8.6}$$

Letting $R(\mathfrak{u}) = (r_n)_{n=1}^\infty$ and $R^\ell(\mathfrak{t}) = \left(r'_n\right)_{n=1}^\infty$, it follows from Definition 8.5 that

$$\mathbb{E}^{\rho,\langle x,q_0\rangle}_{M^\ell}\left(\mathsf{L}^\natural\right) \quad = \quad \mathbb{E}^{\rho,\langle x,q_0\rangle}_{M^\ell}\left(\limsup_{n\to\infty} r'_n\right) \quad = \quad \mathbb{E}^{\rho,\langle x,q_0\rangle}_{M^\ell}\left(\limsup_{n\to\infty} \sum_{k=\max(1,n-\ell+1)}^{n} r_k\gamma^{n-k}\right).$$

Given the equivalence of the probability measures that we have established, conditional upon Equation (8.6), it follows that

$$\mathbb{E}^{\pi,x}_M\left(\mathsf{W}_\ell\mathsf{P}^\natural_\gamma\right) = \mathbb{E}^{\rho,\langle x,q_0\rangle}_{M^\ell}\left(\mathsf{L}^\natural\right).$$

In turn, this implies that

$$\mathrm{Val}_M\left(\mathsf{W}_\ell\mathsf{P}^\natural_\gamma, x\right) = \mathrm{Val}_{M^\ell}\left(\mathsf{L}^\natural, \langle x, q_0\rangle\right). \qquad \square$$

Combining Theorem 8.8 with the results of [CDH09] and [CH07], which establish for every finite MDP that (i) there exist optimal deterministic stationary policies for the payoffs $\mathsf{L}^\sharp$ and $\mathsf{L}^\flat$ and (ii) the upper and lower limit values can be computed in polynomial time, we obtain the following characterization of the strategic and computational complexity of window past-discounted optimization.

Corollary 8.6

Suppose that γ is a discount factor and ℓ is a window length. Over an MDP M with a state set of cardinality k the following statements hold.

(1) There is a deterministic $O\left(k^{\ell}\right)$-memory policy that is optimal for $\mathsf{W}_{\ell}\mathsf{P}_{\gamma}^{\natural}$.

(2) For any state x, the value $\mathrm{Val}_M\left(\mathsf{W}_{\ell}\mathsf{P}_{\gamma}^{\natural}, x\right)$ can be computed in polynomial time when ℓ is fixed and in exponential time when ℓ is part of the problem input.

We now provide a lower bound on the window length needed to approximate a past-discounted value by a window past-discounted value to within a given arbitrary tolerance.

Theorem 8.9

Let $M = \langle X, A, P, R\rangle$ be an MDP and suppose that $r^{\sharp} = \max_{x,y\in X} R(x,y)$ and $r^{\flat} = \min_{x,y\in X} R(x,y)$. For any $\epsilon \geq 0$, if

$$\ell \geq \log_{\gamma}\left(1 - \frac{r^{\sharp} - \epsilon(1-\gamma)}{r^{\flat}}\right) + 1,$$

then

$$\mathrm{Val}_M\left(\mathsf{P}_{\gamma}^{\natural}, x\right) - \mathrm{Val}_M\left(\mathsf{W}_{\ell}\mathsf{P}_{\gamma}^{\natural}, x\right) \leq \epsilon.$$

Proof. Applying the definitions of past-discounted and window past-discounted payoffs yields the equations

$$\mathrm{Val}_M\left(\mathsf{P}_{\gamma}^{\sharp}, x\right) = \sup_{\pi\in\Pi_M} \mathbb{E}_M^{\pi,x}\left(\limsup_{n\to\infty} \sum_{k=1}^{n} r_k \gamma^{n-k}\right)$$

and

$$\mathrm{Val}_M\left(\mathsf{W}_{\ell}\mathsf{P}_{\gamma}^{\natural}, x\right) = \sup_{\rho\in\Pi_M} \mathbb{E}_M^{\rho,x}\left(\limsup_{n\to\infty} \sum_{k=\max(1,n-\ell+1)}^{n} r_k \gamma^{n-k}\right).$$

In this form, it is clear that these values can be bounded as

$$\frac{r^{\flat}}{1-\gamma} \leq \mathrm{Val}_M\left(\mathsf{P}_{\gamma}^{\natural}, x\right) \leq \frac{r^{\sharp}}{1-\gamma}$$

and

$$\frac{r^{\flat}\left(1-\gamma^{\ell-1}\right)}{1-\gamma} \leq \mathrm{Val}_M\left(\mathsf{W}_{\ell}\mathsf{P}_{\gamma}^{\natural}, x\right) \leq \frac{r^{\sharp}\left(1-\gamma^{\ell-1}\right)}{1-\gamma}.$$

Consequently, their difference may be bounded as

$$\mathrm{Val}_M\left(\mathsf{P}_{\gamma}^{\natural}, x\right) - \mathrm{Val}_M\left(\mathsf{W}_{\ell}\mathsf{P}_{\gamma}^{\natural}, x\right) \leq \frac{r^{\sharp}}{1-\gamma} - \frac{r^{\flat}\left(1-\gamma^{\ell-1}\right)}{1-\gamma}.$$

Therefore, the inequality

$$\frac{r^\sharp}{1-\gamma} - \frac{r^\flat\left(1-\gamma^{\ell-1}\right)}{1-\gamma} \leq \epsilon \tag{8.7}$$

implies that

$$\mathrm{Val}_M\left(\mathsf{P}_\gamma^\natural, x\right) - \mathrm{Val}_M\left(\mathsf{W}_\ell \mathsf{P}_\gamma^\natural, x\right) \leq \epsilon.$$

Treating the inequality (8.7) as an equation and solving for ℓ yields

$$\ell = \log_\gamma\left(1 - \frac{r^\sharp - \epsilon(1-\gamma)}{r^\flat}\right) + 1. \tag{8.8}$$

Since $\mathrm{Val}_M\left(\mathsf{W}_\ell \mathsf{P}_\gamma^\natural, x\right)$ increases with ℓ, the upper bound of ϵ on the difference of values of the two payoffs holds as long as ℓ is bounded below by the right-hand side of Equation (8.8). □

Part III

Foundations Of Regular Transformations

In this part, we turn our attention to the theory of regular transformations, and develop a number of theoretical results around regular transductions and the transducers that model them. The motivation behind this portion of the manuscript revolves around enabling an extension of our work on regular Markov decision processes from Chapter 5. In particular, the guiding aspiration is to define a notion of *transition-regular Markov decision processes*, in which state transitions correspond to *regular transductions* as opposed to rational transductions. In the following three chapters, we

- introduce *ω-regular relations* as an extension of regular transductions to infinite strings
- extend the framework of regular model checking to *transition-regular model checking*, in which transitions are regular transductions instead of rational transductions,
- establish and prove correctness of a construction for composing regular transductions modeled as streaming string transducers.

In aggregate, these contributions bring the theory of regular transformations closer to being able to support our vision of transition-regular decision processes.

Chapter 9

Regular Relations Of Infinite Strings

In this chapter, we extend the notion of regular transductions to the setting of ω-strings and ω-languages. We present two models of transductions of infinite strings, one machine based and the other logic based. The main result is Theorem 9.1, which establishes the equivalence of these models and supports our use of the tile "regular" for these transductions.

9.1 Imperative Characterization

First, we propose a transducer model that extends streaming string transducers to operate over infinite strings .

Definition 9.1 – ω-NSST Syntax

A nondeterministic streaming string transducer T over ω-strings is given by a tuple $(\Sigma, \Gamma, Q, I, \mathsf{Acc}, \delta, f, X, A)$, where

- Σ and Γ are finite input and output alphabets;
- Q is a finite set of states;
- $I \subseteq Q$ is a set of initial states;
- Acc is an acceptance condition;
- X is a finite set of string variables;
- A is a finite set of variable update functions of type $X \to (X \cup \Gamma)^*$;
- δ is a transition function of type $(Q \times \Sigma) \to 2^{A \times Q}$;
- $f \in X$ is an append-only output variable.

Such a machine is deterministic (a ω-DSST) if $|\delta(q,a)| = 1$, for all states $q \in Q$ and symbols $a \in \Sigma$, and $|I| = 1$; it is nondeterministic otherwise. On each transition $q_k \xrightarrow[\alpha_k]{\sigma_k} q_{k+1}$, the transducer changes

state and applies the update α_k to each variable of X in parallel. An ω-NSST is *copyless* if every variable in X occurs at most once in the image $\operatorname{im}\alpha$ of every update $\alpha \in A$. Alternately stated, an update $\alpha \in A$ is copyless if the string $\alpha(x_0)\alpha(x_1)...\alpha(x_{n-1})$ has at most one occurrence of each $x \in X$, and an ω-NSST is copyless if all of its updates are copyless.

A run of an ω-NSST on an infinite string $a_1 a_2 ... \in \Sigma^\omega$ is an infinite sequence of states and transitions $q_0 \xrightarrow[\alpha_0]{\sigma_0} q_1 \xrightarrow[\alpha_1]{\sigma_1} ...$ where $q_0 \in I$ and $(q_{k+1}, \alpha_k) \in \delta(q_k, a_k)$ for all $k \in \mathbb{N}$. Let $\mathsf{Runs}_T(w)$ be the set of all runs in T, given input w. An update function $\alpha : X \to (X \cup \Gamma)^*$ can easily be extended to $\alpha : (X \cup \Gamma)^* \to (X \cup \Gamma)^*$ such that $\alpha(w) = \varepsilon$ if $w = \varepsilon$, $\alpha(w) = b\alpha(w')$ if $w = bw'$, and $\alpha(x)\alpha(w')$ if $w = xw'$. The effect of two updates $\alpha_1, \alpha_2 \in A$ in sequence can be *summarized* by the function composition $\alpha_1 \circ \alpha_2$; likewise a sequence of updates of arbitrary length would be summarized by $\alpha_0 \circ \alpha_1 \circ ... \circ \alpha_{n-1}$. For notational convenience, we often omit the hats when the extension is clear from context. Notice that if all updates in a sequence of compositions are copyless, then so is the entire summary.

A valuation is a function $X \to \Gamma^*$ mapping each variable to a string value. The initial valuation $\mathcal{V}_\varepsilon$ of all variables is the empty string ε. A valuation is well-defined after any finite prefix r_n of a run r and is computed as a composition of updates occurring on this prefix: $\mathcal{V}_{r_n} = \mathcal{V}_\epsilon \circ \alpha_0 \circ \alpha_1 \circ ... \circ \alpha_{n-1}$. The output $\mathcal{V}_r(f) = \lim_{n\to\infty} \mathcal{V}_{r_n}(f)$ of T on r is well-defined only if r is accepted by T. Since the output variable f is only ever appended to and never prepended, this limit exists and is an ω-string whenever r is accepted, otherwise we set $\mathcal{V}_r(f) = \bot$.

Definition 9.2 – ω-NSST Semantics

The relation $[\![T]\!]$ realized by an ω-NSST T is defined by the equation

$$[\![T]\!] = \{(w, \mathcal{V}_r(f)) : r \in \mathsf{Runs}_T(w)\}.$$

An ω-NSST T is *functional* if for every w the image $[\![T]\!](w)$ has cardinality at most 1.

Example 9.1

Let Σ be a finite alphabet and # be a special separator not in Σ. For $u, v \in \Sigma^*$, we say that $v \preceq u$ if v is a suffix of u. Consider a relation $R_{\overleftarrow{u}u}$ that transforms strings in $(A \cup \{\#\})^\omega$ such that each maximal #-free finite substring u occurring in the input string is transformed into $\overleftarrow{v}v$ for some suffix v of u. Formally, $R_{\overleftarrow{u}u}$ is defined as

$$\{(u_1\#...\#u_n\#w, \overleftarrow{v_1}v_1\#...\#\overleftarrow{v_n}v_n\#w) : w \in \Sigma^\omega \text{ and } \forall k \leq n.\, u_k, v_k \in \Sigma^* \wedge v_l \preceq u_k\}$$
$$\cup$$
$$\{(u_1\#u_2\#..., \overleftarrow{v_1}v_1\#\overleftarrow{v_2}v_2\#...) : \forall k.\, u_k, v_k \in \Sigma^* \wedge v_k \preceq u_k\},$$

and can be implemented as an ω-NSST with Büchi acceptance condition (accepting states are visited infinitely often for accepting strings) as shown in Figure 9.1.

We consider both Büchi and Muller acceptance conditions for ω-NSSTs and reference these classes of machines by the initialisms NBT and NMT (DBT and DMT for their deterministic versions), respectively. For a run $r \in \mathsf{Runs}_T(w)$, let $\mathrm{Inf}(r) \subseteq Q$ denote the set of states visited infinitely often.

(1) A *Büchi acceptance condition* is given by a set of states $F \subseteq Q$ and is interpreted such that a NBT is defined on an input $w \in \Sigma^\omega$ if there exists a run $r \in \mathsf{Runs}_T(w)$ for which $\mathrm{Inf}(r) \cap F \neq \emptyset$.

(2) A *Muller acceptance condition* is given as a set of sets $\mathbb{F} = \{F_0, ..., F_n\} \subseteq 2^Q$, interpreted such that a NMT is defined on input $w \in \Sigma^\omega$ if there exists a run $r \in \mathsf{Runs}_T(w)$ for which $\mathrm{Inf}(r) \in \mathbb{F}$.

We combine and generalize results in the literature on NSSTs and ω-DSSTs to propose the computational model of nondeterministic streaming ωstring transducers (ω-NSST) capturing regular relations of ωstrings.

The equivalence of NBT-definable and NMT-definable relations follows from a straightforward application of the equivalence of nondeterministic Büchi automata and nondeterministic Muller automata. Equivalence of these acceptance conditions in transducers allows us to switch between them whenever convenient.

Proposition 9.1

A relation is NBT definable if, and only if, it is NMT definable.

Observe that DMTs and functional NMTs, both of which were introduced in [AFT12], have a slightly different output mechanism, which is defined as a function $\Omega : 2^Q \to X^*$ such that the output

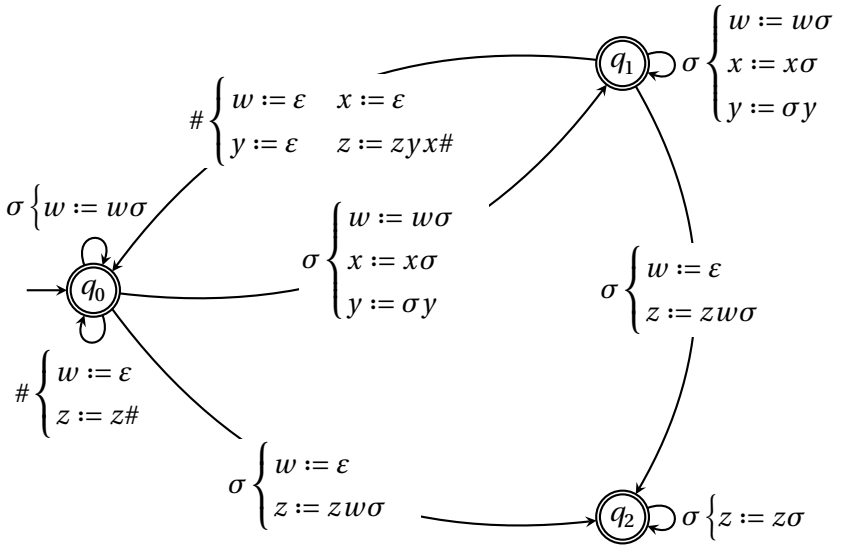


Figure 9.1: An ω-NSST implementing the relation $R_{\overline{u}u}$ from Example 9.1.

string $\Omega(Q')$ is copyless and of the form $x_1 ... x_n$, for all $Q' \subseteq Q$ for which $\Omega(Q') \neq \bot$. Furthermore, there is the condition that if $q, q' \in Q'$ and $a \in \Sigma$ such that $(\alpha, q') \in \delta(q, a)$, then (1) $\alpha(x_k) = x_k$ for all $k < n$ and (2) $\alpha(x_n) = x_n w$ for some $w \in (X \cup \Gamma)^*$.

In contrast, our definition has a unique append-only output variable $f \in X$. However, our model with the Muller acceptance is as expressive as that studied in [AFT12]. One can use nondeterminism to guess a position in the input after which states in a Muller accepting set Q' will be visited infinitely often. The output function can be defined by guessing a Muller set, and keeping an extra variable for the output. Upon making the guess, it will move the contents of $x_1 ... x_n$ to the variable f and make a transition to a copy $T_{Q'}$ of the transducer where Acc = $\{Q'\}$. If any state outside the set Q' is visited, or the variables $x_1 ..., x_{n-1}$ are updated, or the variable f is assigned in non-appending fashion, then $T_{Q'}$ makes a transition to a rejecting sink state. [AFT12] showed the equivalence of functional NMT with DMT. This implies that the transductions definable using functional NMTs or functional NBTs (in our definition) are precisely those definable by ω-DMSOT.

9.2 Declarative Characterization

In this section, we present a declarative characterization obtained by extending monadic second-order logic transducers to operate on ω-strings.

Monadic Second-Order Logic On Strings (MSOS)

Syntax: Strings may be viewed as ordered structures encoded over a signature $\{(\sigma)_{\sigma\in\Sigma}, <\}$. Formulas in MSOS are defined using a countable set of first-order variables $x, y, z, \ldots$ that range over $\mathbb{N}$ and a countable set of second-order variables $X, Y, Z, \ldots$ that range over subsets of $\mathbb{N}$. The grammar for well-formed MSOS formulas is

$$\phi ::= \exists X.\, \phi \mid \exists x.\, \phi \mid \phi \vee \phi \mid \neg\phi \mid \sigma(x) \mid x < y \mid x \in X$$

Variables that are not bound to any quantifier are called free variables.

Semantics: An MSOS formula φ with free variables $FV(\varphi)$ may be modeled by a pair $\langle w, v\rangle$, where $w \in \Sigma^*$ is a string and $v : FV(\varphi) \to \mathbb{N} \cup 2^{\mathbb{N}}$ a valuation of the free variables, according to the following semantics:

- $\langle w, v\rangle \models \sigma(x)$ iff $w_{v(x)} = \sigma$;
- $\langle w, v\rangle \models x < y$ iff $v(x) < v(y)$;
- $\langle w, v\rangle \models x \in X$ iff $v(x) \in v(X)$;
- $\langle w, v\rangle \models \exists x.\phi$ iff there exists $n \in \mathbb{N}$ such that $n \leq |w|$ and $\langle w, v \cup \{x \rightarrowtail n\}\rangle \models \phi$;
- $\langle w, v\rangle \models \exists X.\phi$ iff there exists $N \subseteq \mathbb{N}$ such that $\langle w, v \cup \{X \rightarrowtail N\}\rangle \models \phi$;
- $\langle w, v\rangle \models \neg\phi$ iff $\langle w, v\rangle \not\models \phi$;
- $\langle w, v\rangle \models \phi \vee \psi$ iff $\langle w, v\rangle \models \phi$ or $\langle w, v\rangle \models \psi$.

The language of a formula φ is defined as the set

$$L(\varphi) = \left\{ w \in \Sigma^* \;:\; \exists v : FV(\varphi) \to \mathbb{N} \cup 2^{\mathbb{N}}.\, \langle w, v\rangle \models \varphi \right\}.$$

Monadic Second-Order Transducers of Infinite Strings

MSO transducers are particular specifications in this logic that define transformations between strings. Intuitively, each such transducer copies each input string some fixed number of times and treats the positions in each copy as nodes in a graph, which are then relabeled and rearranged in accordance with the formulas of the transducer to produce an output.

Definition 9.3 – ω-DMSOT

A deterministic MSO ω-string transducer (ω-DMSOT) is a tuple $\langle \Sigma, \Gamma, \mathsf{domain}, N, \Phi, \Psi \rangle$, where

- Σ is a finite input alphabet;
- Γ is a finite output alphabet;
- $N = \{1, ..., n\}$ is a set of copy indices;
- domain is an MSO sentence that defines an input language;
- $\Phi = \left\{ \phi_\gamma^n(x) : n \in N \text{ and } \gamma \in \Gamma \right\}$ is a set of *node formulas*, each with a single free first-order variable, that specify labels for positions in output strings;
- $\Psi = \{\psi^{n,m}(x,y) : n, m \in N\}$ is a set of *edge formulas*, each with two free first-order variables, that specify adjacency of positions in output strings.

A ω-DMSOT operates over N disjoint copies of the string graph of an input. Each formula ϕ_b^n has a single free variable and should be interpreted such that if a position satisfies ϕ_b^n, then that position will be labeled by the symbol b in the n^{th} disjoint string graph comprising the output. Each formula $\psi^{(n,m)}$ has two free variables and a satisfying pair of indices indicates that there is a link between the former index in copy n and the latter index in copy m.

Nondeterminism is introduced through additional set variables $X_1, ..., X_k$ called *parameters*. Fixing a valuation—sets of positions of the input graph satisfying the domain formula—of these parameters determines an output graph, just as in the deterministic case. Each possible valuation may result in a different output graph for the same input graph, and thus nondeterminism arises from the choice of valuation.

Definition 9.4 – ω-NMSOTs

A nondeterministic MSO ω-string transducer (ω-NMSOT) with k free second-order variables $\boldsymbol{X} = \{X_1, ..., X_k\}$ is given as a tuple $\langle \Sigma, \Gamma, \mathsf{domain}(\boldsymbol{X}), N, \Phi(\boldsymbol{X}), \Psi(\boldsymbol{X}) \rangle$, where

- Σ is a finite input alphabet;
- Γ is a finite output alphabet;
- $N = \{1, ..., n\}$ is a set of copy indices;
- $\mathsf{domain}(\boldsymbol{X})$ is an MSO formula with k free second-order variables that defines an input language;
- $\Phi(\boldsymbol{X}) = \left\{ \phi_\gamma^n(x, \boldsymbol{X}) : n \in N \text{ and } \gamma \in \Gamma \right\}$ is a set of *node formulas*, each with a single free first-order variable and k free second-order variables, that specify labels for positions in output strings;
- $\Psi(\boldsymbol{X}) = \{ \psi^{n,m}(x, y, \boldsymbol{X}) : n, m \in N \}$ is a set of *edge formulas*, each with two free first-order variables and k free second-order variables, that specify adjacency of positions in output strings.

Example 9.2

We now describe a ω-NMSOT capturing the relation given in Example 9.1. Set $\Sigma = \{a, b, \#\} = \Gamma$, $N = \{1, 2\}$, and consider a single parameter $\boldsymbol{X}_1 = \{X_1\}$. The domain of the relation is simply Σ^ω, so we omit the formula. For all symbols $\gamma \in \Gamma$ and copy indices $n \in N$, the node formulas labels each position with the same symbol as the corresponding position in the input string: $\phi_\gamma^n(x, X_1) = \gamma(x)$. We omit formal specifications of the edge formulas (which can be found in the extended version of this work [Dav+21a]) and describe them informally. The formula for edges from copy 1 to copy 1 connects adjacent non-# positions that belong to X_1 in the reverse order. The formula for edges from copy 1 to copy 2 connects non-# positions to themselves when the predecessor position is not in X_1. The formula for edges from copy 2 to copy 2 links the right-most sequence of positions in X_1 that precede a # symbol and also connect all those positions coming after the final # if required. Finally, the formula for edges from copy 2 to copy 1 links # symbols to the last position in X_1 left of the next #.
Two possible outputs from the relation of Example 9.1 are displayed in Figure 9.2 which shows how the above ω-NMSOT constructs an output string for two different valuations of X_1. A 1 in the blue (resp. red) row signifies that the position at that column is in X_1, while a 0 indicates it is not in X_1.

9.3 Equivalence Of Imperative & Declarative Models

[AD11] showed that relations over finite strings definable by nondeterministic MSO transducers coincide with those definable by nondeterministic streaming string transducers. We generalize this result by proving that a relation is definable by an ω-NMSOT if, and only if, it is definable by an ω-NSST. We provide symmetric arguments to connect ω-NSST, ω-DSST and ω-NMSOT, ω-DMSOT, resulting in a simple proof.

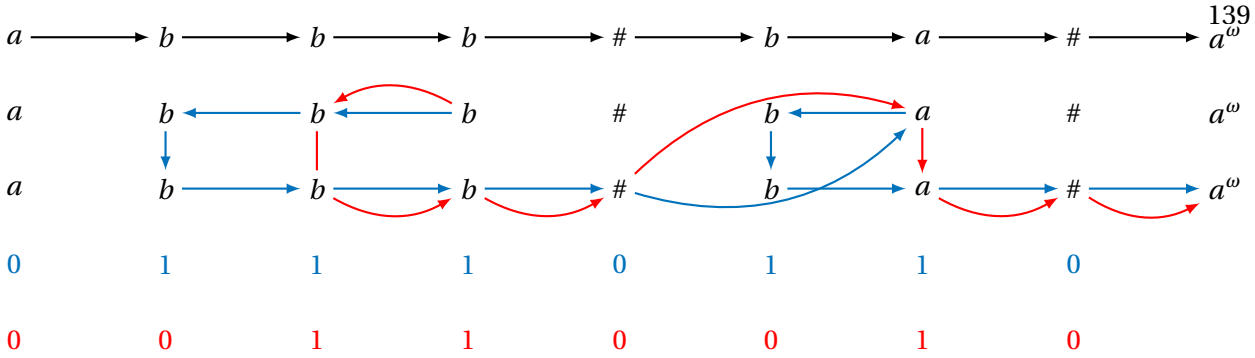


Figure 9.2: Two possible outputs of the relation given in Example 9.1 constructed according to the ω-NMSOT from Example 9.2.

Our arguments use the concept of a relabeling relation, following [EH99; EH01]. A relation $\rho : \Sigma^\omega \times \Gamma^\omega$ is a *relabeling*, if there exists another relation $\rho' : \Sigma \times \Gamma$ such that $(\sigma w_1, \gamma w_2) \in \rho$ iff $(\sigma, \gamma) \in \rho'$ and $(w_1, w_2) \in \rho$. In other words, ρ is obtained by lifting the letter-to-letter relation ρ', in a straightforward manner, to ω-strings. Let $\mathrm{Let}(\rho)$ denote the letter to letter relation $\rho' : \Sigma \times \Gamma$ corresponding to ρ and let ReLab be the set of all such relabelings.

Theorem 9.1 – ω-NMSOT = ω-NSST

The class of relations captured by ω-NMSOTs coincides exactly with the class of relations captured by ω-NSSTs.

The proof of Theorem 9.1 proceeds in two stages. In the first part (Lemma 9.1), we show that every ω-NSST is equivalent to the composition of a nondeterministic relabeling and a ω-DSST. In the second part (Lemma 9.2), we show that every ω-NMSOT is equivalent to the composition of a nondeterministic relabeling and a ω-DMSOT. These two lemmas, in conjunction with the equivalence of DMTs and functional NMTs [AFT12], allow us to equate these two models of transformation via a simple assignment.

Lemma 9.1

$$\omega\text{-NSST} = \omega\text{-DSST} \circ \text{ReLab}$$

Proof. We first show ω-DSST $\circ$ ReLab $\subseteq$ ω-NSST by proving that for every DMT $T = \langle \Gamma, \Lambda, Q, I, \mathbb{F}, \delta, f, X, A \rangle$ and nondeterministic relabeling $\rho : \Sigma^\omega \times \Gamma^\omega$, there is a NMT $T' = \left\langle \Sigma, \Lambda, Q, I, \mathbb{F}, \delta_\rho, f, X, A \right\rangle$ such that

$[\![T']\!] = [\![T]\!] \circ \rho$. As indicated by the tuple given to specify T', the only distinct components between the two machines are their input alphabets and their transition functions δ and δ_ρ. The latter is given as

$$\delta_\rho(q,\sigma) = \bigcup_{\langle\sigma\gamma\rangle \in \mathsf{Let}(\rho)} \delta(q,\gamma).$$

The nondeterminism of ρ is therefore captured in δ_ρ. This results in a unique run through T', for every possible relabeling of inputs for T. Since the remaining pieces of T are untouched in the process of constructing T', it is clear that $[\![T']\!] = [\![T]\!] \circ \rho$.

What remains to be shown is the inclusion ω-NSST $\subseteq$ ω-DSST $\circ$ ReLab: for any NMT T, there exists a DMT T' and a nondeterministic relabeling ρ such that $[\![T]\!] = [\![T']\!] \circ \rho$. From $T = \langle \Sigma, \Gamma, Q, I, \mathbb{F}, \delta, f, X, A\rangle$, we can construct a nondeterministic, letter-to-letter relation $\rho' : \Sigma \times (A \times Q)$ as

$$\rho' = \{\langle \sigma, \langle \alpha, q' \rangle \rangle \ : \ \langle \alpha, q' \rangle \in \delta(q,\sigma)\}.$$

Now let $\rho : \Sigma^\omega \times (A \times Q)^\omega$ be the extension of ρ' as described previously. The relation ρ contains the set of all possible runs through T for any possible input in Σ^ω.

Next, we construct a DMT $T' = \left\langle A \times Q, \Gamma, Q, I, \mathbb{F}, \delta_\rho, f, X, A \right\rangle$ with transition function

$$\delta_\rho\big(q, \langle \alpha, q' \rangle\big) = \{\langle \alpha, q' \rangle \ : \ \exists \sigma \in \Sigma. \, \langle \alpha, q' \rangle \in \delta(q,\sigma)\}.$$

Consequently, T' retains only the pairs in ρ which correspond to valid runs T and encodes them as ω-strings over the alphabet $Q \times A$. The DMT T' then simply follows the instructions encoded in its input and thereby simulates only legitimate runs through T. Thus, we may conclude that $[\![T]\!] = [\![T']\!] \circ \rho$. □

Lemma 9.2

ω-NMSOT = ω-DMSOT $\circ$ ReLab.

Proof. We begin by showing the inclusion ω-NMSOT $\subseteq$ ω-DMSOT $\circ$ ReLab: for any ω-NMSOT T, there exists an ω-DMSOT T' and a relabeling ρ such that $[\![T]\!] = [\![T']\!] \circ \rho$. Nondeterministic choice in T is determined by the choice of assignment to free variables in $\boldsymbol{X}_k$. Alternatively, the job of facilitating nondeterminism can be placed upon a relabeling relation, thereby allowing us to remove the parameter

variables. Define a letter-to-letter relation $\rho' : \Sigma \times (A \times \{0,1\}^k)$ as

$$\rho' = \left\{\langle \sigma, \langle \sigma, \boldsymbol{b}\rangle\rangle \;:\; \sigma \in \Sigma \text{ and } \boldsymbol{b} \in \{0,1\}^k\right\},$$

and let the relabeling $\rho : \Sigma^\omega \times \left(\Sigma \times \{0,1\}^k\right)^\omega$ be its extension. This relabeling essentially gives us a new alphabet such that each symbol from Σ is tagged with encodings of its membership status for each set parameter from $\boldsymbol{X}_k$. Now, we can construct an ω-DMSOT T' that is identical to T, apart from two distinctions. Firstly, T' is deterministic (*i.e.* it has no free set variables), and every occurrence of a subformula $x \in X_i$ in T is replaced by a subformula

$$\bigvee_{\boldsymbol{b} \in \{0,1\}^k \wedge \boldsymbol{b}_i = 1} \langle \sigma, \boldsymbol{b}\rangle(x)$$

in T'. As a result of this encoding, the equality $[\![T]\!] = [\![T']\!] \circ \rho$ holds.

The converse inclusion, ω-DMSOT $\circ$ ReLab$\subseteq$ ω-NMSOT, is much simpler. Every relabeling ρ in ReLab is ω-NMSOT definable: consider $\rho' = \mathsf{Let}(\rho) : \Sigma \times \Gamma$. The ω-NMSOT specifying ρ is similar to identity/copy, except that here we have that the output label is γ iff the input label is σ and $\langle \sigma, \gamma\rangle \in \rho'$. This can be implemented using second-order variables X_γ for all $\gamma \in \Gamma$. Let $\boldsymbol{X}_\Gamma$ represent this set. Only a single copy is required to produce the output. Node formulas are given by

$$\phi^1_\gamma(x, \boldsymbol{X}_\Gamma) = \bigvee_{\sigma \in \Sigma} \bigvee_{\langle \sigma, \gamma\rangle \in \rho'} \Big(\sigma(x) \wedge x \in X_\gamma\Big),$$

and the edge formulas by

$$\psi^{1,1}(x, y, \boldsymbol{X}_\Gamma) = x < y.$$

It is known that ω-NMSOT are closed under composition [CE12]. Thus, we conclude that any composition of a nondeterministic relabeling and a ω-DMSOT is definable by a ω-NMSOT and that ω-DMSOT $\circ$ ReLab$\subseteq$ ω-NMSOT. □

In conjunction Lemmas 9.1 and 9.2 along with the results of [AFT12] allow us to write the following equation, thereby proving Theorem 9.1:

$$\omega\text{-NMSOT} \quad = \quad \omega\text{-DMSOT} \circ \text{ReLab} \quad = \quad \text{DMT} \circ \text{ReLab} \quad = \quad \text{NMT} \quad = \quad \omega\text{-NSST}.$$

Chapter 10

Transition-Regular Model Checking

This chapter generalizes the regular model checking approach—discussed in Chapter 5—so that transition relations can be expressed using *regular relations* over infinite strings. We call this extension *transition-regular model checking*.

Let T^k denote the k-fold composition of a transition relation T with itself. Let T^* denote the transitive closure of T. Suppose that $\mathcal{I}$ and $\mathcal{B}$ are (ω-)regular languages representing sets of states in some system that are initial, and unsafe, respectively. Given a generic transition relation T which captures the dynamics of the system, the *regular model checking problem* asks to decide whether any element of $\mathcal{B}$ is reachable from any element of $\mathcal{I}$ via repeated applications of T. In precise terms, the regular model checking problem asks to decide whether the equation $[\![T^*]\!](\mathcal{I}) \cap \mathcal{B} = \emptyset$ holds. Bounded model checking, in this setting, asks to decide, given $n \in \mathbb{N}$, whether $[\![T^k]\!](\mathcal{I}) \cap \mathcal{B} = \emptyset$ holds, for all $k \leq n$. Unbounded model checking is undecidable, so we focus on bounded model checking.

When T is a rational relation, its image is always a regular language, and this permits the approach of iteratively applying T from $\mathcal{I}$ and checking whether this set intersects with $\mathcal{B}$ by standard automata-theoretic methods. If T is a regular relation, on the other hand, its image may not be a regular language, and we must iteratively compute compositions of T with itself and test whether these compositions enter the $\mathcal{B}$ language. The operation of composition is the focus of Chapter 11. To enable verifying safety properties as we have just described, we establish decidability of the *type checking problem* for ω-regular transductions.

10.1 Type Checking

To facilitate regular model checking with ω-regular relations, we require an algorithm for deciding the type checking problem for ω-regular relations.

Definition 10.1 – Type Checking Problem

The type checking problem for ω-regular relations asks to determine the inclusion of the domain and range of an ω-regular relation with respect to a pair of ω-regular languages.

INPUTS:

(1) an ω-regular relation θ;
(2) a pair of ω-regular languages L_1 and L_2.

DECIDE: whether the inclusions $L_1 \subseteq \operatorname{dom}\theta$ and $[\![\theta]\!](L_1) \subseteq L_2$ hold.

The following theorem establishes that the type checking problem for ω-regular relations is decidable, and, modulo certain assumptions regarding the encoding of inputs, belongs to the complexity class PSPACE

Theorem 10.1

The type checking problem for ω-regular relations, represented as ω-NSSTs, is in PSPACE.

Proof. Suppose that $T = \langle \Sigma, \Gamma, Q, I, F, \delta, f, X, A \rangle$ is an NBT and $L_1 \subseteq \Sigma^\omega$ and $L_2 \subseteq \Gamma^\omega$ are ω-regular languages, encoded, respectively, as deterministic Muller automata (DMA) M_1 and M_2. We first check whether T is defined for all ω-strings $w \in L_1$, *i.e.* whether $L_1 \subseteq \operatorname{dom} T$. A nondeterministic Büchi automaton (NBA) $\mathcal{B}$ that recognizes the domain of T can be constructed in linear time by ignoring variables and output mechanism. The inclusion $L_1 \subseteq \operatorname{dom} T$ can be decided in PSPACE by checking emptiness of $M_1' \cap \mathcal{B}$ where M_1' is the NBA equivalent to M_1 and $\mathcal{B}$ is the NBA representing the complement language of $\operatorname{dom} T$. It is known that an NBA can be constructed from a DMA with exponential blowup in the number of states [Bok18]. A complement automaton can be constructed for an NBA with exponential increase in the number of states as well [Bok18]. Hence, $\mathcal{B}$ has exponentially many states relative to T and M_1. Intersection of M_1' and $\mathcal{B}$ is a standard product construction with a flag so that both M_1' and $\mathcal{B}$ visit good states infinitely often. Thus, the intersection NBA $M_1' \cap \mathcal{B}$ has exponentially many states

relative to T and M_1. Thanks to the fact that emptiness of NBA can be checked in NLOGSPACE [Bok18], the emptiness of this product automaton, can be decided in NPSPACE = PSPACE.

We now assume that T is well-defined on L_1 and construct a nondeterministic Muller automaton (NMA) $\mathcal{A}$ such that the language of $\mathcal{A}$ is defined as

$$\{w \in L_1 : \exists w' \in [\![T]\!](w).\ w' \notin L_2\}.$$

Next, we construct a DMA $\overline{M_2}$ for $\overline{L_2}$ by complementing the Acc set. The automaton $\mathcal{A}$ simulates M_1, T and $\overline{M_2}$ in parallel. Next, we construct an NMT T' corresponding to the NBT T in order to homogenize the acceptance condition across these machines. Let us fix the definition for all three machines:

(1) $M_1 = \langle \Sigma, Q_1, p_0, \mathbb{F}_1, \delta_1 \rangle$;

(2) $T' = \langle \Sigma, \Gamma, Q, I, \mathbb{F}', \delta, f, X, A \rangle$;

(3) $\overline{M_2} = \langle \Gamma, Q_2, r_0, \mathbb{F}_2, \delta_2 \rangle$.

The NMA $\mathcal{A}$ is defined as the product of M_1 and T' (without the output mechanism), and it stores a state summary map—*i.e.* the effect of running current valuation of each variable starting from all states of $\overline{M_2}$—in each of its own states. Formally, the states of $\mathcal{A}$ comprise a finite subset of $Q_1 \times Q \times (Q_2 \times X \to Q_2 \cup \{\bot\})$. A state $\langle q, p, g \rangle$ with $g(r, x) = r'$ represents that, starting from state r, if we read the current value of variable x, then we reach state r'. If $g(r, x) = \bot$, it indicates that there is no run on valuation of x starting from r. This information can be updated along the run of $\mathcal{A}$. For instance, if a transition of T updates x to $aybx$, then the summary map g is updated to g' such that

$$g'(r, x) = g(\delta_2(g(\delta_2(r, a), y), b), x),$$

summarizing the effect of reading $x = aybx$ in $\overline{M_2}$ starting from state r.

The set of states of $\mathcal{A}$ is $Q_{\mathcal{A}} = Q_1 \times Q \times (Q_2 \times X \to Q_2 \cup \{\bot\})$, in which Q_1, Q, and Q_2 represent the state sets of M_1, T', and $\overline{M_2}$, respectively. The transition relation $\delta_{\mathcal{A}}$ is defined such that $\langle q', p', g' \rangle \in \delta_{\mathcal{A}}(\langle q, p, g \rangle, \sigma)$ iff

(1) $\delta_1(q, \sigma) = q'$;

(2) $\langle \alpha, p' \rangle \in \delta_1(p, \sigma)$;

(3) $g'(r, x) = r'$ and $\delta_2^*\big(r, \mathcal{V}_{\alpha(x)}\big) = r'$, for all $x \in X$ and $r \in Q_2$.

Initial states are the product of initial states, *i.e.* a set $I_{\mathcal{A}} = \{\langle q_0, p_0, r_0 \rangle : q_0 \in I\}$. The Muller accepting set of $\mathcal{A}$ is defined as the collection of all $P \subseteq Q_{\mathcal{A}}$ such that (1) $\pi_1(P) \in \mathbb{F}_1$, (2) $\pi_2(P) \in \mathbb{F}$, and (3) $(\pi_3(P))(r_0, f) \in \mathbb{F}_2$, where π_i is the i^{th} projection. The size of NMA $\mathcal{A}$ is exponential in the number variables of T, polynomial in the number of states of M_1 and T. Thanks to the fact that emptiness of an NMA can be determined in NLOGSPACE, emptiness of $\mathcal{A}$ having exponential states in the inputs T, M_1 and M_2, can be decided in NPSPACE and thus, by Savitch's Theorem [Sav70], also in PSPACE. □

Remark 10.1

Note that in the above proof, the complexity of the decision problem may differ if the input automata are given with different acceptance conditions.

10.2 Bounded Model Checking & Beyond

Since regular relations are definable in MSO, they are closed under sequential composition. In combination with Theorems 9.1 and 10.1, this establishes the necessary conditions for bounded regular model checking with regular relations to be possible.

Corollary 10.1

Bounded model checking with regular relations is decidable.

Despite the fact that unbounded regular model checking is undecidable, bounded regular model checking provides a refutation procedure. That is, it allows us to search for a witness for proving the system unsafe. Unfortunately, we cannot use bounded model checking of this kind to decide if the overall system does satisfy the desired property. On the other hand, we identify several special cases of the problem which permit the safety of the system to be verified in finite time.

Functional Fixed Points.

The first partial approach to unbounded model checking applies when T is functional, *i.e.* $[\![T]\!]$ is a function, and relies on the following theorem.

Theorem 10.2 – Alur, Filiot, and Trivedi [AFT12]

- Functionality is decidable for ω-NSSTs.
- Equivalence is decidable for functional ω-NSSTs.

At every step of the bounded regular model checking procedure, one can check if T^k is functional, if T^{k+1} is functional, and if $[\![T^k]\!] = [\![T^{k+1}]\!]$. If these three conditions hold, then, for all $m \geq 0$, we have that $[\![T^k]\!] = [\![T^{k+m}]\!]$. When this occurs and $[\![T^k]\!](L) \subseteq \overline{B}$ holds, it follows that $[\![T^k]\!] = [\![T^*]\!]$. As a result, the inclusion $[\![T^*]\!](L) \subseteq \overline{B}$ holds, which implies that $[\![T^*]\!](L) \cap B = \emptyset$.

Note that T^k can be functional even when T is not. To see this, consider a non-functional ω-NSST T such that $[\![T]\!](a^\omega) = \{b^\omega, c^\omega\}$ and $[\![T]\!](b^\omega) = [\![T]\!](c^\omega) = d^\omega$. If $a^\omega \in L$ and $|[\![T]\!](w)| = 1$ for every other input w and $a^\omega \notin \operatorname{im} T$, then T^2 is functional.

Inductive Invariants

An alternative approach involves showing that $[\![T]\!]$ satisfies some inductive invariant. Select, as a candidate invariant, a regular or ω-regular language L which is contained in the set of safe states $L \subseteq \overline{B}$. Now, L provides a witness to the unbounded safety of the system if the following pair of conditions are met: $L \subseteq L$ and $[\![T]\!](L) \subseteq L$. Together, these conditions imply that $[\![T^*]\!](L) \subseteq L$, and in combination with the assumption that $L \subseteq \overline{B}$ this yields that $[\![T^*]\!](L) \cap B = \emptyset$. The necessary inclusions can be formulated as instances of the type checking problem, and so, given an appropriately chosen inductive invariant in the form of an ω-regular language, the global safety of such a system may be verified in polynomial space. This method is easily generalized by searching for k-inductive invariants: ω-regular languages for which there is a $k \in \mathbb{N}$ such that $[\![T^k]\!](L) \subseteq L$.

Chapter 11

Composing Copyless Streaming String Transducers

While regular transductions are known to be closed under composition thanks to the declarative characterization based on MSO logic, converting an MSO transducer into an imperative model like an SST incurs a non-elementary blowup in the size of the representation. To avoid this concern in the context of transition-regular model checking, it would be preferable to make use of direct methods to compose imperative models of regular transductions. This chapter develops a technique for directly composing streaming string transducers.

Streaming string transducers (SSTs) are machine models for transformations between formal languages. Upon reading a symbol, an SST changes state and updates a finite set of string variables, which are initially set to the empty string ε, using concatenation expressions, such as $x := aybzc$, made up of variables y, z and symbols a, b, c from a designated output alphabet. The final output of the transducer is determined by a similar combination of variables and symbols, depending on the final state reached after reading an input word. Deterministic SSTs (DSSTs) [AČ10; AČ11] model functions over strings, while nondeterministic SSTs (NSSTs) [AD11] are interpreted as relations.

(a) Copyless SST S.

(b) Copyful SST T.

Figure 11.1: Representative examples of copyless and copyful SSTs.

An SST is *copyless* if each variable is used at most once across all variable updates occurring on a single transition. The copyless restriction prohibits updates involving assignments such as

- $\begin{cases} x := aybyc \\ y := x \end{cases}$, in which two copies of y flow into x;
- $\begin{cases} x := y \\ y := y \end{cases}$, where one copy of y flows into each of the variables x and y.

An SST that is not copyless is called *copyful* [FR17; FR21]. Figure 11.1a shows an example, S, of a copyless SST implementing the function $w \mapsto ww$. Figure 11.1b provides an example, T, of a copyful SST that implements the mapping $a^k \mapsto a^{2^{k+1}}$, which is not definable by any copyless SST. In Figure 11.1a, σ denotes an arbitrary symbol of an arbitrary alphabet.

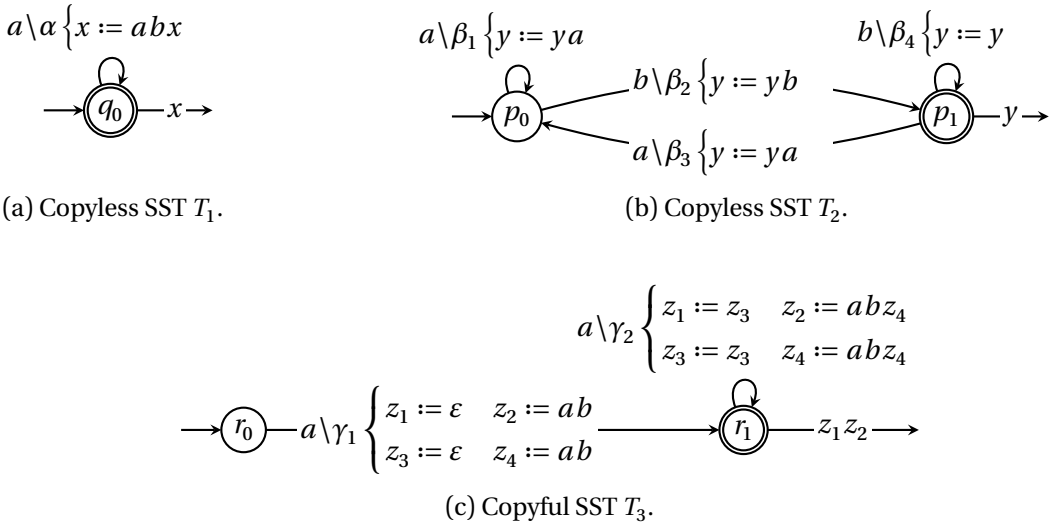
The behavior of SSTs with respect to variable copying is particularly significant when it comes to characterizing the expressive power of the model. Copyful SSTs, as evidenced by Figure 11.1, can model transductions having output strings of lengths exponential in the lengths of corresponding input strings. Copyless SSTs, on the other hand, have outputs with lengths at most linear in the lengths of the corresponding inputs. As shown by Alur, et al. [AČ10; AD11], the transductions definable by copyless SSTs coincide with those definable by monadic second-order logic graph transducers (MSOTs) [Cou92; Cou94; CE12]. The assumption that variable updates are copyless is crucial for this equivalence, because the linear upper bound on lengths of output strings is a fundamental property of MSOTs.

As MSOTs are closed under composition[1] , it follows from their correspondence that the functions and relations definable by SSTs are also closed under composition. Effective procedures are given in [AČ10; AD11] to construct a copyless SST that realizes the composition of two copyless SSTs given as input. Joost Engelfriet observed[2] , however, that these constructions can produce transducers with copyful assignments. Consider, for instance, the three deterministic SSTs shown in Figure 11.2[3] , where T_3 is the composition of T_1 and T_2, constructed according to the methods provided by [AČ10; AD11]. The transducer T_1 implements the function $a^n \mapsto (ab)^n$, while the transducer T_2 models the mapping $a^{n_1}b^{m_1}...a^{n_k}b^{m_k} \mapsto a^{n_1}b...a^{n_k}b$. Notice that both T_1 and T_2 are copyless, yet there is a transition in T_3

[1] *cf.* Proposition 3.2 of [Cou92; Cou94] and Theorem 7.14 of [CE12].

[2] via a private correspondence with the authors of [AD11]

[3] An edge labeled by $a\backslash\alpha$ indicates that the machine transitions from the edge's source state to its target state upon reading symbol a and applies the assignment α to its set of variables.

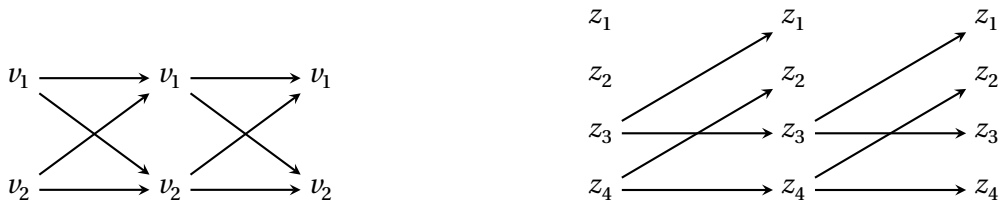


(a) Copyless SST T_1.

(b) Copyless SST T_2.

(c) Copyful SST T_3.

Figure 11.2: Two copyless SSTs T_1, T_2 and a copyless SST T_3 implementing their composition.



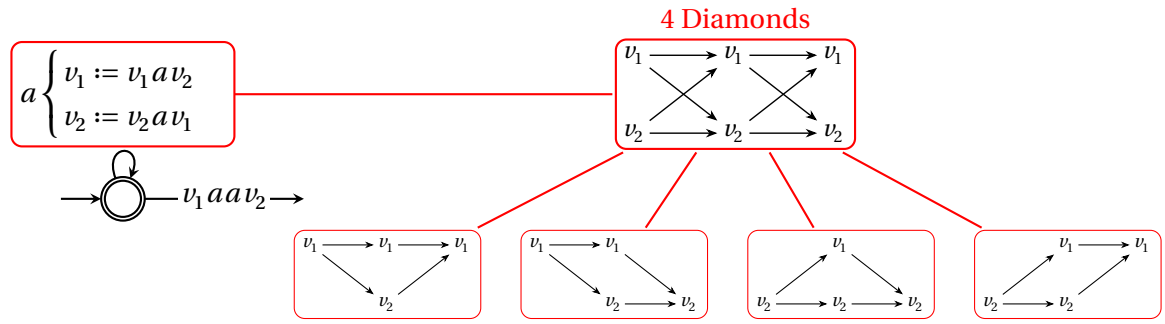
(a) Two-step flow graph in T.

(b) Flow graph for $\gamma_2 \circ \gamma_2$ in T_3.

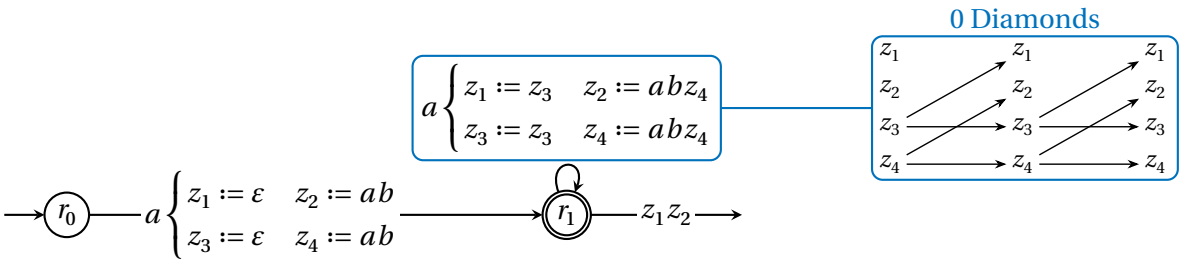
Figure 11.3: A pair of two-step flow graphs from T and T_3.

with a copyful assignment γ_2. On the other hand, notice how none of the variables copied by γ_2 are ever combined in any later assignment.

Directed acyclic graphs are a useful tool for representing the dynamics of variable copying in SSTs, facilitating both analysis and visualization. The graphs depicted in Figure 11.3, for instance, provide convenient encodings of how information flows between variables of SSTs over the course of a computation. In these illustrations, an edge $x \to y$ corresponds to a transition that involves an assignment to variable y in which x occurs. Figure 11.3a shows how variables flow into other variables during two consecutive applications of the single assignment of T, while Figure 11.3b shows this for two consecutive applications of the assignment γ_2 of T_3. Despite γ_2 being copyful, the diagram makes clear that there is no possibility of two copies of z_3 or z_4 converging and contributing to a common variable



(a) Diamonds in T.



(b) Absence of the diamonds in T_3.

at a later point in time. We call copyful SSTs exhibiting this type of behavior *diamond-free*. Intuitively, an SST is diamond-free if there is at most one path between any two vertices in the assignment graph induced by any computation. The diamond-free restriction, while weaker than the copyless restriction, preserves its semantic intent: that outputs are linear in the lengths of their corresponding inputs.

Related Work. The diamond-free property has been considered in related literature [LMT19] studying determinization of finitely-ambiguous cost register automata. While our notion of diamond-freeness essentially coincides with the notion considered in that work, our results are independent and apply in a distinct context.

An alternative restriction on copying behavior in SSTs, called *bounded-copy*, is also considered in related work [AFT12]. An SST is bounded-copy, if in the assignment graph for any computation, there are finitely many paths between any two vertices. For any non-negative integer k, a bounded copy SST is called k-copy if there are at most k paths between any two vertices in the assignment graph induced by a computation. From this definition, it is clear that copyless SSTs correspond to 0-copy SSTs and diamond-free SSTs correspond to 1-copy SSTs. Note, however, that [AFT12] considers deterministic SSTs

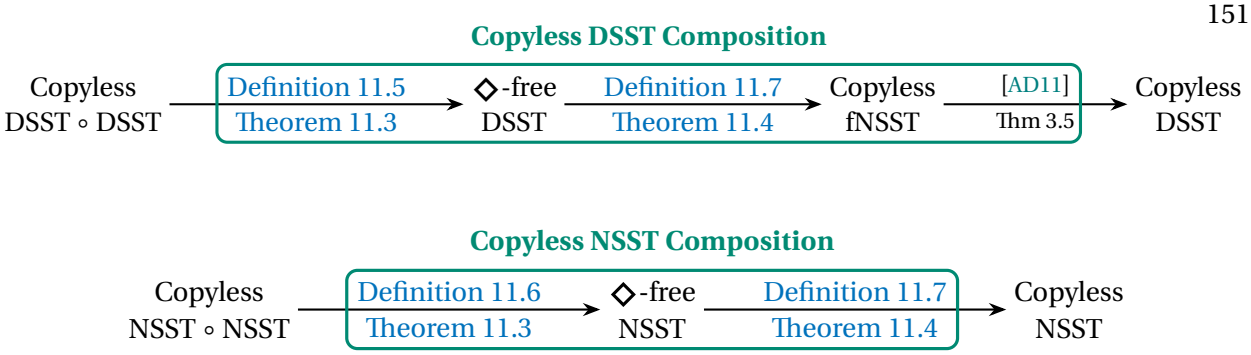


Figure 11.5: Schematic of our approach to composing copyless SSTs.

while the present work considers nondeterministic SSTs. Furthermore, the methods we use to convert diamond-free NSSTs into copyless NSSTs are simpler and distinct from those employed in [AFT12] to convert bounded-copy DSSTs into copyless DSSTs.

Streaming String Transducers

Before we jump into the technical definitions, it should be noted that we use a different, but equivalent, formulation of SSTs than was used in Chapter 9. The SSTs considered in this chapter operate over finite strings. Additionally, they use an output function, rather than an output variable, which is closer to the original formulation presented by Alur, et al.

Definition 11.1 – SST Syntax

A streaming string transducer T is given by a tuple $\langle \Sigma, \Lambda, Q, q_0, X, A, \delta, F \rangle$, where

- Σ is a finite input alphabet,
- Λ is a finite output alphabet,
- Q is a finite set of states,
- $q_0 \in Q$ is a distinguished initial state,
- X is a finite set of string variables,
- A is a finite set of assignment maps with type $X \to (\Lambda \cup X)^*$ for updating variables,
- $\delta \subseteq Q \times \Sigma \times A \times Q$ is a transition relation;
- $F : Q \to (\Lambda \cup X)^*$ is a partial[a] output function.

[a] By "partial", we mean that there may be states on which F is undefined.

An SST is deterministic (DSST), if for every pair $\langle q, s \rangle \in Q \times \Sigma$ there is exactly one transition

$\langle q, s, \alpha, q' \rangle \in \delta$, and is nondeterministic (NSST) otherwise.

An assignment α of type $X \to (\Lambda \cup X)^*$ can naturally be extended to a morphism $(\Lambda \cup X)^* \to (\Lambda \cup X)^*$ such that $\alpha(s) = s$, for $s \in \Lambda$ and $\alpha\left(\prod_{k=1}^{n} w_k\right) = \prod_{k=1}^{n} \alpha(w_k)$, for words in $(\Lambda \cup X)^*$. When viewed in this manner, assignments can be composed in sequence, and we set $\bigcirc_{k=1}^{n} \alpha_k = \alpha_n \circ \alpha_{n-1} \circ \ldots \circ \alpha_1$ as the notation for iterated composition.

The *shape* of a string $w_0 \prod_{k=1}^{n} x_k w_k \in (X \cup \Lambda)^*$ is a summary $\square x_1 \square x_2 \ldots \square x_n \square$ of the string's topology, which ignores details of the variable-free substrings occurring between variables. Here, the symbol $\square$ simply indicates the presence of a substring $w_k \in \Lambda^*$. Such shapes are entirely determined by the order of the variables occurring in the original string.

A run of an SST on an input word $w = w_1 \ldots w_n \in \Sigma^*$ is a finite sequence of transitions $q_0 \xrightarrow[\alpha_1]{w_1} \ldots \xrightarrow[\alpha_n]{w_n} q_n$ such that $\langle q_{k-1}, w_k, \alpha_k, q_k \rangle \in \delta$ for all $1 \leq k \leq n$. For a run ρ, define the *valuation* $\mathcal{V}_\rho : (X \cup \Lambda)^* \to \Lambda^*$ as $\mathrm{Er}_X \circ \bigcirc_{k=1}^{n} \alpha_k$.

Definition 11.2 – SST Semantics

An SST T specifies a relation $[\![T]\!] \subseteq \Sigma^* \times \Lambda^*$ where

$$[\![T]\!] = \left\{ \left\langle w, \mathcal{V}_\rho \circ F(q_n) \right\rangle : \text{exists run } \rho \text{ in } T \text{ on } w \text{ ending in } q_n \in \operatorname{dom} F \right\}.$$

A *functional* NSST is one for which every element of $\operatorname{dom} [\![T]\!]$ maps to a unique element of $\operatorname{im} [\![T]\!]$.

Copying Variables

A string $w \in (\Sigma_1 \cup \Sigma_2)^*$ is copyless in Σ_1 if each element of Σ_1 occurs at most once in w, *i.e.* when $\max_{s \in \Sigma_1} |w|_s \leq 1$. We write $[\Sigma_1]$ for the set of copyless strings over Σ_1 and $[\Sigma_1, \Sigma_2]$ for set of strings over $(\Sigma_1 \cup \Sigma_2)^*$ that are copyless in Σ_1.

An assignment $\alpha \in A$ is copyless if the string $\prod_{k=1}^{|X|} \alpha(x_k)$ is copyless in X, making it an element of $[X, \Lambda]$. It is straightforward to observe that any composition of copyless assignments is also copyless. We say an SST is copyless if, for all transitions $\langle q, s, \alpha, q' \rangle \in \delta$, the assignment α is copyless. An SST is copyful if it has at least one copyful assignment.

The following proposition characterizes the cardinality of the set of all copyless strings over a

finite alphabet in terms of the size of that alphabet. It will be used to establish bounds on the size of composite SSTs.

Proposition 11.1

If $|X| = n$ and $n \geq 2$, then $|[X]| = \lfloor en! \rfloor$.

Proof. Strings in $[X]$ can be any length less than or equal to n, so counting the number of elements amounts to counting the number of ways to choose—without repetition—and order k elements from X for $0 \leq k \leq n$. Thus,

$$|[X]| \quad = \quad \sum_{k=0}^{n} \frac{n!}{(n-k)!} \quad = \quad n! \sum_{k=0}^{n} \frac{1}{(n-k)!} \quad = \quad n! \sum_{k=0}^{n} \frac{1}{k!}.$$

By Taylor's theorem [Rud76],

$$e = \sum_{k=0}^{n} \frac{1}{k!} + \frac{e^a}{(n+1)!},$$

holds for some $a \in (0,1)$, and so we obtain the equation

$$|[X]| \quad = \quad n! \left(e - \frac{e^a}{(n+1)!} \right) \quad = \quad en! - \frac{e^a}{n+1}.$$

Since the map $a \mapsto e^a$ is increasing on the unit interval, the subtrahend of the above difference may be bounded as

$$\frac{1}{n+1} \quad \leq \quad \frac{e^a}{n+1} \quad \leq \quad \frac{e}{n+1}.$$

Furthermore, the assumption that $n \geq 2$ implies that $\frac{e^a}{n+1} < 1$. Finally, $[X]$ is a finite set and thus has cardinality in the positive integers, so we conclude that

$$|[X]| = \lfloor en! \rfloor. \qquad \square$$

Flow Graphs & Diamonds

The copying behavior of an SST is characterized by *flow graphs* associated to its assignments.

Definition 11.3 – Flow Graph

Let $\alpha = \alpha_1 ... \alpha_n$ be a word over A^*. The flow graph of α is a directed acyclic graph $G(\alpha) = \langle V(\alpha), E(\alpha) \rangle$ such that

- the set of vertices $V(\alpha) = \biguplus_{k=1}^{n+1} X_k$ is made up of $n+1$ disjoint copies of X and
- $\langle x, y \rangle \in E(\alpha)$ if there exists $1 \leq k \leq n$ such that $x \in X_k$, $y \in X_{k+1}$, and $x \in \alpha_k(y)$.

Definition 11.4 – Diamond

Let $\alpha = \alpha_1 ... \alpha_n$ be a word over A^*, and let $G(\alpha)$ be its flow graph. A diamond is a subgraph of $G(\alpha)$, consisting of two distinct paths from a source vertex $s \in X_i$ to a target vertex $t \in X_j$, with $i < j$.

An assignment α is *diamond-free* if $|\alpha(x)|_y \leq 1$ holds for all $x, y \in X$ or, equivalently, if $G(\alpha)$ has no double edges. Copyless assignments are diamond-free, but diamond-free assignments are not necessarily copyless. The flow graph $G(\rho)$ of a run $\rho = q_0 \xrightarrow[\alpha_1]{w_1} \dots \xrightarrow[\alpha_n]{w_n} q_n$ is identified with the flow graph $G(\alpha_1 ... \alpha_n)$ of its sequence of assignments. A run ρ is diamond-free if there is at most one path between any two vertices in $G(\rho)$. Note that runs and SSTs may fail to be diamond-free even if all assignments in A are diamond-free. An SST is diamond-free if all of its runs are diamond-free.

11.1 Composition Construction

This section presents a modified and elaborated treatment of the composition constructions of [AČ10] and [AD11]. Our departure from the original versions is centered around two aspects:

(1) we impose a strict indexing of the variables of the composite SST that them to be partitioned based on their roles in the process;

(2) we provide explicit detailed definitions, in the form of recursive higher-order functions, of the operators used to produce the composite SST.

The indexing scheme in combination with the recursively defined operators allow us to establish the diamond-free property in an almost type-directed manner.

We begin by presenting the construction under the assumption that the given SSTs are deterministic in Section 11.1. Then, in Section 11.1, we describe the necessary adaptations to the procedure for

NSSTs. In Section 11.1.1, we prove correctness of the modified constructions and establish bounds on the size of the composite SSTs. Lastly, in Section 11.2, we prove that the composite transducers must be diamond-free.

DSST Composition

Fix copyless DSSTs $T_2^1 = \langle \Sigma_1, \Sigma_2, Q, q_0, X, A, \delta_2^1, F_2^1 \rangle$ and $T_3^2 = \langle \Sigma_2, \Sigma_3, P, p_0, Y, B, \delta_3^2, F_3^2 \rangle$ as inputs. The DSST $T_3^1 = \langle \Sigma_1, \Sigma_3, R, r_0, \mathcal{Z}, C, \delta_3^1, F_3^1 \rangle$ will be the composite transducer. The set of variables of T_3^1 is written as $\mathcal{Z}$ to allow the use of uppercase Z for individual variables, which makes our index notation more readable.

The main idea underlying the composition constructions is to simulate the state transitions of T_2^1 while keeping a record of what T_3^2 does when reading each possible variable in X from each possible state in P. Since T_2^1 builds its output piecewise, these records are critical. By storing the actions of T_3^2 for each possible eventuality, it is possible to then chain together those actions which correspond to the run of T_3^2 on the output of T_2^1. In particular, the composite transducer incorporates two types of *summaries* as finite maps in the states of T_3^1.

- *State summaries* are maps $f : (P \times X) \to P$ that record the state in T_3^2 that is reached after reading the contents of a variable x, for every possible combination of starting states in $p \in P$ of T_3^2 and variables $x \in X$ of T_2^1.
- *Shape summaries* are maps $g : (P \times X \times Y) \to [Y \cup \mathcal{Z}]$ that store the shape of each variable $y \in Y$ after reading a variable $x \in X$ from a starting point $p \in P$, for every possible combination of parameters.

Since their domains and ranges are finite, each summary is finitely representable and there are finitely many summaries of each type. We write $\mathbb{F}$ for the set of all functions with type $(P \times X) \to P$ and $\mathbb{G}$ for the set of all functions of type $(P \times X \times Y) \to [Y \cup \mathcal{Z}]$. The variables $\mathcal{Z} = \left\{ Z_{y,0}^{p,x}, Z_{y,1}^{p,x} : \langle p, x, y \rangle \in P \times X \times Y \right\}$ of T_3^1 are used by the shape summary maps to store variable-free substrings of shapes.

We now define the higher-order operators that will be used to explicitly define the composition constructions. For the remainder of this section, we set the following conventions.

- $\mathbf{x}$ represents strings from $[X, \Sigma_2]$.
- $\mathbf{y}$ denotes strings from $[Y \cup \mathcal{Z}, \Sigma_3]$ and $[Y, \Sigma_3]$.
- $\mathbf{z}$ represents strings from $[\mathcal{Z}, \Sigma_3]$.
- g_x^p is shorthand for the map $y \mapsto g(p, x, y)$, where $g \in \mathbb{G}$.

State Transition Summarizer. The map $\mathcal{F} : P \times [X, \Sigma_2] \times \mathbb{F} \to P$ is responsible for updating destination states in T_3^2 based on updated variables in T_2^1 and is defined as

$$\mathcal{F}(p, \mathbf{x}, f) = \begin{cases} p & \text{if } \mathbf{x} = \varepsilon \\ \mathcal{F}\big(p', \mathbf{x}', f\big) & \text{if } \mathbf{x} = s\,\mathbf{x}' \text{ and } \langle p, s, \alpha, p' \rangle \in \delta_3^2 \\ \mathcal{F}\big(f(p, x), \mathbf{x}', f\big) & \text{if } \mathbf{x} = x\,\mathbf{x}'. \end{cases} \tag{11.1}$$

Assignment Summarizer. The processes that create shape summaries and assignments share a dependency on the map $\mathcal{G} : P \times [X, \Sigma_2] \times \mathbb{F} \times \mathbb{G} \to (Y \to [Y \cup \mathcal{Z}, \Sigma_3])$ defined as

$$\mathcal{G}(p, \mathbf{x}, f, g) = \begin{cases} \mathrm{Id}_Y & \text{if } \mathbf{x} = \varepsilon \\ \beta \circ \mathcal{G}\big(p', \mathbf{x}', f, g\big) & \text{if } \mathbf{x} = s\,\mathbf{x}' \text{ and } \langle p, s, \beta, p' \rangle \in \delta_3^2 \\ g_x^p \circ \mathcal{G}\big(f(p, x), \mathbf{x}', f, g\big) & \text{if } \mathbf{x} = x\,\mathbf{x}'. \end{cases} \tag{11.2}$$

Shape Generator. The map $\mathcal{S} : P \times X \times (Y \to [Y \cup \mathcal{Z}, \Sigma_3]) \to (Y \to [Y \cup \mathcal{Z}])$ computes shape summaries and is given as

$$\mathcal{S}(p, x, h) = y \mapsto \left(\prod_{k=1}^{n} Z_{y_k,0}^{p,x}\, y_k \right) Z_{y,1}^{p,x} \qquad \text{if } h(y) = \mathbf{z}_0 \prod_{k=1}^{n} y_k\, \mathbf{z}_k. \tag{11.3}$$

Assignment Generator. The map $\mathcal{A} : P \times X \times (Y \to [Y \cup \mathcal{Z}, \Sigma_3]) \to (\mathcal{Z} \to [\mathcal{Z}, \Sigma_3])$ computes assignments and is given as

$$\mathcal{A}(p, x, h) = \begin{cases} Z_{y,0}^{p,x} \mapsto \varepsilon & \text{if } y \notin h(y'), \text{ for all } y' \in Y \\ Z_{y,0}^{p,x} \mapsto \mathbf{z} & \text{if } h(y') = \mathbf{z}\,y\,\mathbf{y} \text{ or } y''\,\mathbf{z}\,y \in h(y') \text{ for some } y', y'' \in Y \\ Z_{y,1}^{p,x} \mapsto \mathbf{z} & \text{if } h(y) = \mathbf{z} \text{ or } h(y) = \mathbf{y}\,y'\,\mathbf{z}, \text{ for some } y' \in Y. \end{cases} \tag{11.4}$$

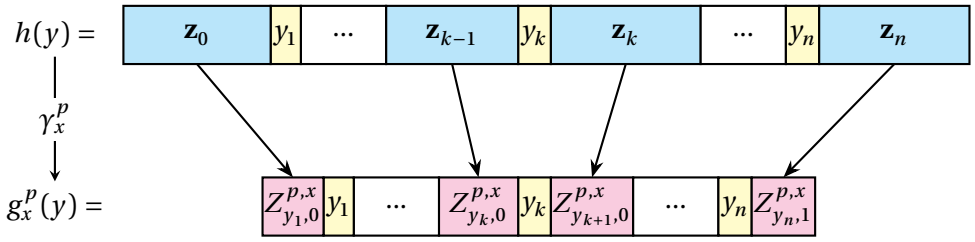


Figure 11.6: An illustration of the relationship between an assignment summary $h = \mathcal{G}(p, \alpha(x), f, g)$ and the corresponding shape $g_x^p = \mathcal{S}_x^p(h)$ and assignment $\gamma_x^p = \mathcal{A}_x^p(h)$.

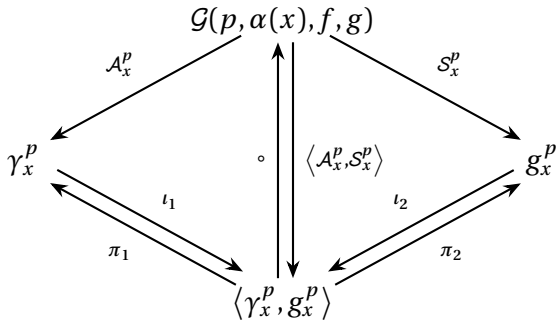
We write $\mathcal{S}_x^p$ and $\mathcal{A}_x^p$, respectively, for the mappings $h \mapsto \mathcal{S}(p, x, h)$ and $h \mapsto \mathcal{A}(p, x, h)$. Letting

$$g_x^p = \mathcal{S}_x^p \circ \mathcal{G}(p, \alpha(x), f, g) \quad \text{and} \quad \gamma_x^p = \mathcal{A}_x^p \circ \mathcal{G}(p, \alpha(x), f, g),$$

the next equation holds for all $y \in Y$:

$$\mathcal{G}(p, \alpha(x), f, g)(y) = \gamma_x^p \circ g_x^p(y). \tag{11.5}$$

Thus, the following diagram commutes, where π_k and ι_k are the k^{th} canonical projection and injection.



This relationship is further illustrated in Figure 11.6.

The following definition formalizes the construction of T_3 from DSSTs T_1 and T_2.

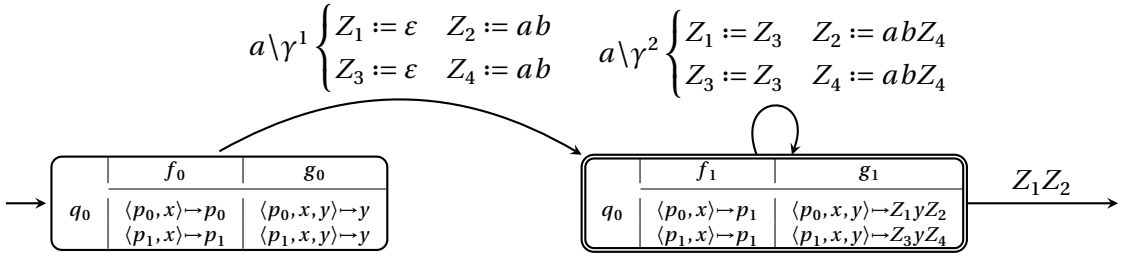


Figure 11.7: A detailed view of internal maps contained in the states of T_3 from Figure 11.2, when constructed according to Definition 11.5. Note that we omit unreachable states.

Definition 11.5 – DSST Composition

Suppose that $T_2^1 = \langle \Sigma_1, \Sigma_2, Q, q_0, X, A, \delta_2^1, F_2^1 \rangle$ and $T_3^2 = \langle \Sigma_2, \Sigma_3, P, p_0, Y, B, \delta_3^2, F_3^2 \rangle$ are an arbitrary pair of copyless DSSTs. Their composition is a DSST $T_3^1 = \langle \Sigma_1, \Sigma_3, R, r_0, \mathcal{Z}, C, \delta_3^1, F_3^1 \rangle$ where

- $R = Q \times \mathbb{F} \times \mathbb{G}$ such that $r_0 = \langle q_0, f_0, g_0 \rangle$ with $f_0(p, x) = p$ and $g_0(p, x, y) = y$;
- $\mathcal{Z} = \left\{ Z_{y,0}^{p,x}, Z_{y,1}^{p,x} : \langle p, x, y \rangle \in P \times X \times Y \right\}$;
- $C = \left\{ \gamma : \langle r, s, \gamma, r' \rangle \in \delta_3^1 \right\}$;
- $\langle \langle q, f, g \rangle, s, \gamma, \langle q', f', g' \rangle \rangle \in \delta_3^1$ if, there exists $\langle q, s, \alpha, q' \rangle \in \delta_2^1$ such that
  (1) $f'(p, x) = \mathcal{F}(p, \alpha(x), f)$;
  (2) $g'(p, x, y) = \mathcal{S}_x^p(\mathcal{G}(p, \alpha(x), f, g))(y)$;
  (3) $\gamma\left(Z_{y,k}^{p,x}\right) = \mathcal{A}_x^p(\mathcal{G}(p, \alpha(x), f, g))\left(Z_{y,k}^{p,x}\right)$,
- $F_3^1(q, f, g) = \mathrm{Er}_Y \circ \mathcal{G}\left(p_0, F_2^1(q), f, g\right) \circ F_3^2 \circ \mathcal{F}\left(p_0, F_2^1(q), f\right)$.

NSST Composition

Definition 11.5 requires some adaptation for NSSTs. One approach to this adaptation might be to generalize the maps $\mathcal{F}$ and $\mathcal{G}$ independently to accommodate nondeterminism. This would involve defining variations of $\mathcal{F}$ and $\mathcal{G}$ having codomains 2^P and $2^{Y \to [Y \cup \mathcal{Z}, \Sigma_3]}$, respectively. Then, one could add transitions in T_3^1 whenever the summaries in the destination state could be obtained via these maps from the summaries in the source state. Doing this, however, could lead to grouping a state summary based on one nondeterministic run in T_3^2 together with a shape summary based on a different nondeterministic run. To see this, imagine that the NSST in Figure 11.8 is T_3^2. Notice that the string aa leads to four distinct states and induces four distinct assignments, but to faithfully simulate this transducer, each destination state should be paired to a unique assignment. For instance, if the state summary points to p_4, then the

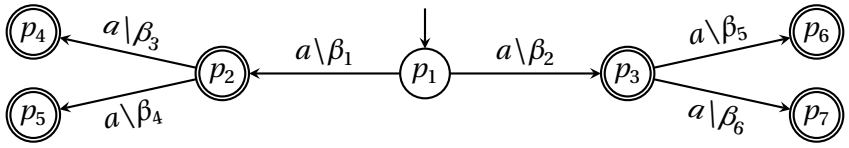


Figure 11.8: An NSST illustrating the need for $\mathcal{H}$.

only shape summary that correctly pairs with this state summary represents the effects of applying the assignment $\beta_3 \circ \beta_1$. Proceeding as we have just described would lead to 3 additional incorrect pairings, and this would result in an incorrect composite transducer.

To avoid such situations, it is necessary to synchronize the computation of state summaries with the computation of shape summaries and assignments. This synchronization is achieved by the map $\mathcal{H} : (P \times [X, \Sigma_2] \times \mathbb{F} \times \mathbb{G}) \to 2^{P \times (Y \to [Y \cup \mathcal{Z}, \Sigma_3])}$ which combines $\mathcal{F}$ and $\mathcal{G}$ in more structured way and is defined as

$$\mathcal{H}(p, \mathbf{x}, f, g) = \begin{cases} \{\langle p, \mathrm{Id}_Y \rangle\} & \text{if } \mathbf{x} = \varepsilon, \\ \bigcup_{\langle p, s, \alpha, p' \rangle \in \delta_3^2} \{\langle p'', \alpha \circ h \rangle : \langle p'', h \rangle \in \mathcal{H}(p', \mathbf{x}', f, g)\} & \text{if } \mathbf{x} = s\,\mathbf{x}', \\ \{\langle p', g_x^p \circ h \rangle : \langle p', h \rangle \in \mathcal{H}(f(p, x), \mathbf{x}', f, g)\} & \text{if } \mathbf{x} = x\,\mathbf{x}'. \end{cases}$$

Definition 11.6 – NSST Composition

Suppose that $T_2^1 = \langle \Sigma_1, \Sigma_2, Q, q_0, X, A, \delta_2^1, F_2^1 \rangle$ and $T_3^2 = \langle \Sigma_2, \Sigma_3, P, p_0, Y, B, \delta_3^2, F_3^2 \rangle$ are an arbitrary pair of copyless NSSTs. Their composition is an NSST $T_3^1 = \langle \Sigma_1, \Sigma_3, R, r_0, \mathcal{Z}, C, \delta_3^1, F_3^1 \rangle$ where

- $R = Q \times \mathbb{F} \times \mathbb{G}$ such that $r_0 = \langle q_0, f_0, g_0 \rangle$ with $f_0(p, x) = p$ and $g_0(p, x, y) = y$;
- $\mathcal{Z} = \left\{ Z_{y,0}^{p,x}, Z_{y,1}^{p,x} : \langle p, x, y \rangle \in P \times X \times Y \right\}$;
- $C = \{\gamma : \langle r, s, \gamma, r' \rangle \in \delta_3^1\}$;
- $\langle \langle q, f, g \rangle, s, \gamma, \langle q', f', g' \rangle \rangle \in \delta_3^1$ if there exist $\langle q, s, \alpha, q' \rangle \in \delta_2^1$ such that
  (1) $\langle p', h \rangle \in \mathcal{H}(p, \alpha(x), f, g)$;
  (2) $f'(p, x) = p'$;
  (3) $g'(p, x, y) = \mathcal{S}_x^p(h)(y)$;
  (4) $\gamma\left(Z_{y,k}^{p,x}\right) = \mathcal{A}_x^p(h)\left(Z_{y,k}^{p,x}\right)$,
- $F_3^1(q, f, g) = \mathrm{Er}_Y \circ h \circ F_3^2(p)$, with $\langle p, h \rangle \in \mathcal{H}(p_0, F_2^1(q), f, g)$ chosen nondeterministically.

Besides accommodating nondeterminism, the use of $\mathcal{H}$ in Definition 11.6 does not change the essential nature of the composition construction. In particular, the methods by which summaries and corresponding assignments are computed are identical in both Definition 11.5 and Definition 11.6. Only the way in which the computed summaries are grouped to form transitions in δ_3^1 differs.

Remark 11.1

In order to minimize bookkeeping and notational complexity, the analyses in Section 11.1.1 and Section 11.2 are carried out for NSSTs using the maps $\mathcal{F}$ and $\mathcal{G}$ rather than $\mathcal{H}$, but with the assumption that the synchronization enforced by $\mathcal{H}$ holds implicitly.

11.1.1 Correctness & Size Bounds

Suppose $\langle s,t\rangle \in [\![T_2^1]\!]$ and $\langle t,u\rangle \in [\![T_3^2]\!]$ such that there are corresponding runs $\rho = q_0 \xrightarrow[\alpha_1]{s_1} \dots \xrightarrow[\alpha_n]{s_n} q_n$ and $\sigma = p_0 \xrightarrow[\beta_1]{t_1} \dots \xrightarrow[\beta_m]{t_m} p_m$ in T_2^1 and T_3^2, respectively. By Definition 11.6, for every transition $\langle q_{k-1}, s_k, \alpha_k, q_k\rangle \in \delta_2^1$ occurring in ρ, there exists a transition $\langle\langle q_{k-1}, f_{k-1}, g_{k-1}\rangle, s_k, \gamma_k, \langle q_k, f_k, g_k\rangle\rangle \in \delta_3^1$. Let τ be the corresponding run $\langle q_0, f_0, g_0\rangle \xrightarrow[\gamma_1]{s_1} \dots \xrightarrow[\gamma_n]{s_n} \langle q_n, f_n, g_n\rangle$ in T_3^1. Also, let $\alpha = \bigcirc_{k=1}^n \alpha_k$ and $\gamma = \bigcirc_{k=1}^n \gamma_k$.

Lemma 11.1

Suppose that $x \in X$ and $p \in P$. If $\sigma(p,x)$ is a run in T_3^2 on input $\mathcal{V}_\rho(x)$, starting from state p and ending in state p', then

$$f_n(p,x) = p'. \tag{11.6}$$

Proof. We proceed by induction on n, the length of ρ and τ.

Base case: *Equation (11.6) holds when* $n = 0$.

If $n = 0$, then $\rho = q_0$ and $\tau = \langle q_0, f_0, g_0\rangle$. Therefore, $\mathcal{V}_\rho(x) = \varepsilon$, for all $x \in X$, and so for all $p \in P$ and $x \in X$, we have $\sigma(p,x) = p = p'$. By Definition 11.6, it holds that $f_0(p,x) = p$.

Inductive case: *If Equation (11.6) holds for* $n-1$, *then it also holds for* n.

By Definition 11.6, $f_n(p,x) = \mathcal{F}(p, \alpha_n(x), f_{n-1})$, for all $\langle p,x\rangle \in P \times X$. Observe that $\mathcal{F}(p, \alpha_n(x), f_{n-1})$ partially simulates the transitions of T_3^2, by reading $\alpha_n(x)$ in sequence and keeping track of the current state. This simulation operates according to δ_3^2 when the next symbol of $\alpha_n(x)$ comes from Σ_2 and

according to f_{n-1} when the next symbol comes from X. In other words, for every state p, $\alpha_n(x)$ induces a sequence of states $p_1, p_2 ..., p_{|\alpha_n(x)|}$, via $\mathcal{F}$, such that $p_1 = p$ and either

(1) $\langle p_k, s, \beta, p_{k+1}\rangle \in \delta_3^2$ for some $\beta \in B$, if the k^{th} symbol of $\alpha_n(x)$ is $s \in \Sigma_2$, or

(2) $f_{n-1}(p_k, x') = p_{k+1}$, if the k^{th} symbol of $\alpha_n(x)$ is $x' \in X$.

The former case is correct by definition, while the latter follows from the inductive hypothesis. Consequently, we arrive at the equation

$$f_n(p,x) = \mathcal{F}(p, \alpha_n(x), f_{n-1}) = p_{|\alpha_n(x)|} = p',$$

and we conclude that correctness of f_{n-1} implies correctness of f_n. □

Lemma 11.2

Suppose that $\gamma : \mathcal{Z} \to [\mathcal{Z}, \Sigma_3]$, $h : Y \to [Y \cup \mathcal{Z}, \Sigma_3]$, and $\phi(a) = a$ for any function ϕ and $a \notin \operatorname{dom}\phi$. For all $Z \in \mathcal{Z}$, the equation $\gamma(Z) = h \circ \gamma(Z) = \gamma \circ h(Z)$ holds.

Proof. Follows from the fact that $\operatorname{dom} h = Y$ is disjoint from $\operatorname{im}\gamma = [\mathcal{Z}, \Sigma_3]$. □

Lemma 11.3

Suppose that $x \in X$, $p \in P$, and $y \in Y$. If $\sigma(p,x)$ is a run in T_3^2 on input $\mathcal{V}_\rho(x)$, starting from state p, then

$$\mathcal{V}_{\sigma(p,x)}(y) = \mathcal{V}_\tau \circ g_n(p,x,y). \tag{11.7}$$

Proof. We proceed by induction on n.

BASE CASE: *Equation (11.7) holds when $n = 0$.*

Because $\sigma(p,x) = p$, we have that $\mathcal{V}_{\sigma(p,x)}(y) = y$, for all $y \in Y$. By Definition 11.6 $g_0(p,x,y) = y$, and so $\mathcal{V}_\tau \circ g_0(p,x,y) = y$. Likewise, $\mathcal{V}_\tau \circ g_0(p,x,y) = \mathcal{V}_{\sigma(p,x)}(y) = y$.

INDUCTIVE CASE: *If Equation (11.7) holds for $n-1$, then it also holds for n.*

Suppose that $\rho' = q_0 \xrightarrow[\alpha_1]{s_1} \ldots \xrightarrow[\alpha_{n-1}]{s_{n-1}} q_{n-1}$, where $\rho = \rho' \xrightarrow[\alpha_n]{s_n} q_n$, and that $\tau' = \langle q_0, f_0, g_0\rangle \xrightarrow[\gamma_1]{s_1} \ldots \xrightarrow[\gamma_{n-1}]{s_{n-1}} \langle q_{n-1}, f_{n-1}, g_{n-1}\rangle$ where $\tau = \tau' \xrightarrow[\gamma_n]{s_n} \langle q_n, f_n, g_n\rangle$. Further, suppose that $\sigma'(p,x)$ is a run in T_3^2 on $\mathcal{V}_{\rho'}(x)$,

starting from state p. Let the last assignment of ρ be $\alpha_n(x) = w_0 x_1 ... x_\ell w_\ell$, and let w_i^j denote the j^{th} symbol in w_i. The run $\sigma(p,x)$ in T_3^2 can be written as

$$\sigma(p,x) = p \xrightarrow[\beta_0^1]{w_0^1} ... p_{|w_0 x_1 ... w_{i-1}|} \xrightarrow[\mathcal{V}_{\sigma'(p_{|w_0 x_1 ... w_{i-1}|}, x_i)}]{x_i} p_{|w_0 x_1 ... w_{i-1} x_i|} \cdots \xrightarrow[\beta_\ell^{|w_\ell|}]{w_\ell^{|w_\ell|}} p_{|\alpha_n(x)|}.$$

Consequently, we obtain the following derivation, which completes the proof.

$$\mathcal{V}_{\sigma(p,x)} = \bigcirc_{k=1}^{|w_0|} \beta_0^k \circ \bigcirc_{i=1}^{\ell} \mathcal{V}_{\sigma'(p_{w_0 x_1 ... w_{i-1}}, x_i)} \circ \bigcirc_{j=1}^{|w_i|} \beta_j^i \tag{11.8}$$

$$= \bigcirc_{k=1}^{|w_0|} \beta_0^k \circ \bigcirc_{i=1}^{\ell} \mathcal{V}_{\tau'} \circ g_{n-1}(p_{w_0 x_1 ... w_{i-1}}, x_i, \cdot) \circ \bigcirc_{j=1}^{|w_i|} \beta_j^i \tag{11.9}$$

$$= \mathcal{V}_{\tau'} \circ \bigcirc_{k=1}^{|w_0|} \beta_0^k \circ \bigcirc_{i=1}^{\ell} g_{n-1}(p_{w_0 x_1 ... w_{i-1}}, x_i, \cdot) \circ \bigcirc_{j=1}^{|w_i|} \beta_j^i \tag{11.10}$$

$$= \mathcal{V}_{\tau'} \circ \mathcal{G}(p, \alpha_n(x), f_{n-1}, g_{n-1}) \tag{11.11}$$

$$= \mathcal{V}_{\tau'} \circ \gamma_n \circ g_n(p,x,\cdot) \tag{11.12}$$

$$= \mathcal{V}_{\tau} \circ g_n(p,x,\cdot) \tag{11.13}$$

From Equation (11.8), we obtain Equation (11.9) from the inductive hypothesis. We get Equation (11.10) by applying Lemma 11.2. From here, Equation (11.11) is obtained from the definition of $\mathcal{G}$ in Equation (11.2). Next, Equation (11.12) follows from Equation (11.5) and the definitions of $\mathcal{S}$ and $\mathcal{A}$. Finally, Equation (11.13) is implied by the definition of valuation. □

Theorem 11.1

If $T_3^1 = T_3^2 \circ T_2^1$ is constructed according to Definition 11.6, then

$$[\![T_3^1]\!] = [\![T_3^2]\!] \circ [\![T_2^1]\!].$$

Proof. Observe the following derivation:

$$[\![T_3^1]\!](s) = \mathcal{V}_\tau \circ F_3^1(\langle q_n, f_n, g_n \rangle) \tag{11.14}$$

$$= \mathcal{V}_\tau \circ \mathcal{G}(p_0, F_2^1(q_n), f_n, g_n) \circ F_3^2 \circ \mathcal{F}(p_0, F_2^1(q_n), f_n) \tag{11.15}$$

$$= \mathcal{V}_\tau \circ \mathcal{G}(p_0, F_2^1(q_n), f_n, g_n) \circ F_3^2(p_m). \tag{11.16}$$

From Equation (11.14), we obtain Equation (11.15) by applying Definition 11.5. Equation (11.16) follows from Equation (11.15) via application of Lemma 11.1. Considering the output of T_2^1 as the value of an

extra variable x_{out}, as in [MP19]. Letting $h = \mathcal{G}(p_0, F_2^1(q_n), f_n, g_n)$, we can rewrite Equation (11.16) as

$$[\![T_3^1]\!](s) = \mathcal{V}_\tau \circ \mathcal{A}_{x_{\text{out}}}^{p_0}(h) \circ \mathcal{S}_{x_{\text{out}}}^{p_0}(h) \circ F_3^2(p_m). \tag{11.17}$$

Since $[\![T_2^1]\!](s) = \mathcal{V}_\rho(x_{\text{out}})$, by definition, applying Lemma 11.3 to Equation (11.17) yields

$$[\![T_3^1]\!](s) = \mathcal{V}_\sigma \circ F_3^2(p_m) = [\![T_3^2]\!](t). \qquad \square$$

Theorem 11.2

Suppose that $T_3^1 = T_3^2 \circ T_2^1$ is constructed according to Definition 11.6. If T_2^1 has k states and ℓ variables and T_3^2 has n states and m variables, then T_3^1 has a state space of size $O(kn^{\ell n}((2\ell nm + m)!)^{\ell nm})$ and $2\ell nm$ variables.

Proof. By Definition 11.5, the state space of R of T_3^1 is comprised by the product $Q \times \mathbb{F} \times \mathbb{G}$ where $\mathbb{F} = (P \times X) \to P$ and $\mathbb{G} = (P \times X \times Y) \to [Y \cup \mathcal{Z}]$. Therefore, we have

$$|\mathbb{F}| = |P|^{|P \times X|} = n^{\ell n} \qquad \text{and} \qquad |\mathbb{G}| = |[Y \cup \mathcal{Z}]|^{|P \times X \times Y|} = |[Y \cup \mathcal{Z}]|^{\ell nm}.$$

It is clear from Definition 11.5 that $|\mathcal{Z}| = 2\ell nm$. Applying Proposition 11.1 yields

$$|[Y \cup \mathcal{Z}]| = \lfloor e(m + 2\ell nm)! \rfloor.$$

As a result, we get that

$$|R| = kn^{\ell n} \lfloor e(m + 2\ell nm)! \rfloor^{\ell nm} = O\Big(kn^{\ell n}((2\ell nm + m)!)^{\ell nm}\Big). \qquad \square$$

11.2 Establishing Diamond-Freeness of Composite Transducers

We define two binary relations over the $\mathcal{Z}$ variables of T_3^1:

$$Z_{y,b}^{p,x} \overset{X}{\sim} Z_{y',b'}^{x',p'} \quad \text{iff} \quad x = x' \qquad \text{and} \qquad Z_{y,b}^{p,x} \equiv Z_{y',b'}^{x',p'} \quad \text{iff} \quad x = x' \ \&\ p = p'.$$

It is easy to see that $\overset{X}{\sim}$ and $\equiv$ are equivalence relations. The relation $\overset{X}{\sim}$ partitions $\mathcal{Z}$ into $|X|$ classes, each of size $2|Y \times P|$. Similarly, the relation $\equiv$ partitions $\mathcal{Z}$ into $|X \times P|$ classes each of size $2|Y|$. We write $Z|_{\underset{\sim}{X}} = x$ to denote that Z belongs to the class indexed by x and $Z|_{\equiv} = \langle p, x \rangle$ to denote that the equivalence class

of Z is indexed by $\langle p, x\rangle$. We next utilize these relations and the structure they impose on $\mathcal{Z}$ to show that the composite transducer is bounded-copy. In the following, α is always an assignment in T_3^1, and Z, Z', Z_1, *etc.* will refer to generic variables from $\mathcal{Z}$.

Lemma 11.4

For any state $\langle q, f, g\rangle \in R$, variable $Z \in \mathcal{Z}$, and $\langle p, x, y\rangle \in P \times X \times Y$, it holds that $|g(p,x,y)|_Z \leq 1$ and if $Z \in g(p,x,y)$, then $Z|_{\equiv} = \langle p, x\rangle$.

Proof. By definition of the initial shape summary g_0 it is clear that $|g_0(p,x,y)|_Z = 0$, regardless of p, x, y, Z. Any other shape summary occurring in T_3^1 is constructed by the map $\mathcal{S}$ which is defined, in Equation (11.3), such that no variable from $\mathcal{Z}$ occurs twice in the image of a single element of the domain. Additionally, it is easy to see from the same definition in Equation (11.3) that any $Z \in \mathcal{S}(p,x,h)(y)$ must be indexed by the pair $\langle p, x\rangle$, invariant of h and y, thus implying that $Z|_{\equiv} = \langle p, x\rangle$. □

Lemma 11.5

Suppose that $\alpha : X \to [X, \Sigma_2]$ is an arbitrary copyless assignment of T_2^1, and that there exist variables $x_1, x_2 \in X$, $y_1, y_2 \in Y$, and $Z_1, Z_2 \in \mathcal{Z}$ such that $Z_1 \in \mathcal{G}(p_1, \alpha(x_1), f, g)(y_1)$ and $Z_2 \in \mathcal{G}(p_2, \alpha(x_2), f, g)(y_2)$. If $x_1 \neq x_2$, then $Z_1 \overset{X}{\nsim} Z_2$.

Proof. If $Z_1 \overset{X}{\nsim} Z_2$, then there must be a unique variable x such that $\mathcal{G}(p_1, \alpha(x_1), f, g)$ uses a shape summary $g_x^{p_3}$ and $\mathcal{G}(p_2, \alpha(x_2), f, g)$ uses a shape summary $g_x^{p_4}$ for some states p_3, p_4. This implies that $x \in \alpha(x_1)$ and $x \in \alpha(x_2)$ which contradicts our assumption that α is copyless. Consequently, we conclude that the pair of inclusions $Z_1 \in \mathcal{G}(p_1, \alpha(x_1), f, g)(y_1)$ and $Z_2 \in \mathcal{G}(p_2, \alpha(x_2), f, g)(y_2)$ imply that $Z_1 \overset{X}{\nsim} Z_2$. □

Lemma 11.6

For any state $\langle q, f, g\rangle \in R$, copyless assignment $\alpha : X \to [X, \Sigma_2]$ of T_2^1, variable $Z \in \mathcal{Z}$, and $\langle p, x, y\rangle \in P \times X \times Y$,

$$1 \geq |\mathcal{G}(p, \alpha(x), f, g)(y)|_Z.$$

Proof. We proceed by induction on $\alpha(x)$.

Base case: $|\mathcal{G}(p, \varepsilon, f, g)|_Z \leq 1$.

By definition of $\mathcal{G}$, in Equation (11.2), it follows that $\mathcal{G}(p,\varepsilon,f,g)(y) = y$ when $\alpha(x) = \varepsilon$. Since no variables from $\mathcal{Z}$ ever occur in such a string, $|\mathcal{G}(p,\varepsilon,f,g)(y)|_Z = 0 \leq 1$.

INDUCTIVE CASE: *If* $|\mathcal{G}(p,\mathbf{x},f,g)(y)|_Z \leq 1$ *and either* $\alpha(x) = \mathbf{x}s$ *or* $\alpha(x) = \mathbf{x}x'$, *then*

$$|\mathcal{G}(p,\alpha(x),f,g)(y)|_Z \leq 1.$$

The disjunction contained in the inductive hypothesis leads us to consider two cases.

(1) Suppose that $\alpha(x) = \mathbf{x}s$. By applying the definition of $\mathcal{G}$ we obtain the equation

$$\mathcal{G}(p,\alpha(x),f,g) = \mathcal{G}(p,\mathbf{x},f,g) \circ \beta,$$

where $(\mathcal{F}(p,\mathbf{x},f),s,\beta,p') \in \delta_3^2$. Since the type of β is $Y \to [Y,\Sigma_3]$, composing this assignment with $\mathcal{G}(p,\mathbf{x},f,g)$ does not introduce or remove any $\mathcal{Z}$ variables from the resulting strings, we get

$$|\mathcal{G}(p,\mathbf{x}s,f,g)(y)|_Z = |\mathcal{G}(p,\mathbf{x},f,g) \circ \beta(y)|_Z = |\mathcal{G}(p,\mathbf{x},f,g)(y)|_Z. \tag{11.18}$$

Assuming the inductive hypothesis, we have $|\mathcal{G}(p,\mathbf{x},f,g)(y)|_Z \leq 1$, which, in combination with Equation (11.18), implies the inequality $|\mathcal{G}(p,\mathbf{x}s,f,g)(y)|_Z \leq 1$.

(2) Otherwise, suppose that $\alpha(x) = \mathbf{x}x'$. Then, applying the definition of $\mathcal{G}$ implies that

$$\mathcal{G}(p,\beta(x),f,g)(y) = \mathcal{G}(p,\mathbf{x},f,g) \circ g(\mathcal{F}(p,\mathbf{x},f),x',y)$$

holds for every $y \in Y$. By assumption, we know that $|\mathcal{G}(p,\mathbf{x},f,g)(y)|_Z \leq 1$. Applying Lemma 11.4, we obtain the inequality $\left|g(\mathcal{F}(p,\mathbf{x},f),x',y)\right|_Z \leq 1$. It is not possible, however, for both $Z \in \mathcal{G}(p,\mathbf{x},f,g)(y)$ and $Z \in g(\mathcal{F}(p,\mathbf{x},f),x',y)$ to hold simultaneously, since this would imply that $|\alpha(x)|_{x'} = 2$, which would thereby contradict the assumption that α is copyless. This mutual exclusivity may be restated quantitatively as

$$|\mathcal{G}(p,\mathbf{x},f,g)(y)|_Z + \left|g(\mathcal{F}(p,\mathbf{x},f),x',y)\right|_Z \leq 1,$$

and this implies that $\left|\mathcal{G}(p,\mathbf{x}x',f,g)(y)\right|_Z \leq 1$. □

Lemma 11.7

For any state $\langle q, f, g\rangle \in R$, copyless assignment $\alpha : X \to [X, \Sigma_2]$, variable $Z \in \mathcal{Z}$, and $\langle p, x\rangle \in P \times X$,

$$1 \geq \sum_Y |\mathcal{G}(p, \alpha(x), f, g)(y)|_Z.$$

Proof. Suppose that $\langle q, s, \alpha, q'\rangle \in \delta_2^1$ is the corresponding transition in T_2^1. If both the inclusions $Z \in \mathcal{G}(p, \alpha(x), f, g)(y_1)$ and $Z \in \mathcal{G}(p, \alpha(x), f, g)(y_2)$ hold, then there must exist $\mathbf{x}, \mathbf{x}' \in (X \cup \Sigma_2)^*$ and $x' \in X$ such that $\mathbf{x}x'\mathbf{x}' = \alpha(x)$ and $Z \in g(\mathcal{F}(p, \mathbf{x}, f), x', y_3)$, for some variable y_3, occurring both $\mathcal{G}(p, \mathbf{x}, f, g)(y_1)$ and $\mathcal{G}(p, \mathbf{x}, f, g)(y_2)$. However, we know that compositions of copyless assignments, such as $\mathcal{G}(p, \mathbf{x}, f, g)$, remain copyless. Thus, if $y_3 \in \mathcal{G}(p, \mathbf{x}, f, g)(y_1)$ and $y_3 \in \mathcal{G}(p, \mathbf{x}, f, g)(y_2)$ both hold, there must exist a copyful assignment in T_3^2. This contradicts our initial assumption that T_3^2 is copyless, so we infer that a single variable Z cannot occur in more than one element of the set $\{\mathcal{G}(p, \alpha(x), f, g)(y) : y \in Y\}$. This fact, in combination with Lemma 11.6, which asserts that Z can occur at most once in any element of this set, allows us to conclude that $\sum_{y\in Y} |\mathcal{G}(p, \alpha(x), f, g)(y)|_Z \leq 1$. □

Lemma 11.8

Suppose that $Z \in \mathcal{Z}$ is a variable of T_3^1 and that $\alpha : X \to [X, \Sigma_2]$ is an assignment of T_2^1. If $Z \in \mathcal{G}(p_1, \alpha(x_1), f, g)(y_1)$ and $Z \in \mathcal{G}(p_2, \alpha(x_2), f, g)(y_2)$, then $x_1 = x_2$ and either (1) $p_1 = p_2$ and $y_1 = y_2$ or (2) $p_1 \neq p_2$.

Proof. Since $Z \overset{X}{\sim} Z$, Lemma 11.5 implies that $x_1 = x_2$, we are left with two cases.

(1) First, suppose that $p_1 = p_2$. If $y_1 \neq y_2$, then $Z \in \mathcal{G}(p, \alpha(x), f, g)(y_1)$ and $Z \in \mathcal{G}(p, \alpha(x), f, g)(y_2)$. This implies the equation

$$|\mathcal{G}(p, \alpha(x), f, g)(y_1)|_Z + |\mathcal{G}(p, \alpha(x), f, g)(y_2)|_Z = 2,$$

and thus contradicts Lemma 11.7. Therefore, it follows that $p_1 = p_2$ implies $y_1 = y_2$.

(2) Otherwise, suppose $p_1 \neq p_2$. If $y_1 = y_2$, then it follows that both $Z \in \mathcal{G}(p_1, \alpha(x), f, g)(y)$ and $Z \in \mathcal{G}(p_2, \alpha(x), f, g)(y)$ hold. Conversely, if $y_1 \neq y_2$, then both $Z \in \mathcal{G}(p_1, \alpha(x), f, g)(y_2)$ and $Z \in \mathcal{G}(p_2, \alpha(x), f, g)(y_1)$ hold. Neither possibility raises a contradiction, so the inequality $p_1 \neq p_2$ is independent of the relationship between y_1 and y_2. □

Lemma 11.9

For any assignment $\gamma : \mathcal{Z} \to [\mathcal{Z}, \Sigma_3]$, if there exist variables $Z, Z_1, Z_2 \in \mathcal{Z}$ such that $Z \in \gamma(Z_1)$ and $Z \in \gamma(Z_2)$, then $Z_1 \overset{X}{\sim} Z_2$ and either $Z_1 = Z_2$ or $Z_1 \not\equiv Z_2$.

Proof. If $Z \in \gamma(Z_1)$ and $Z \in \gamma(Z_2)$, then there exist pairs $\langle p_1, x_1 \rangle$ and $\langle p_2, x_2 \rangle$ such that we have the inclusions $Z \in \mathcal{A}^{p_1}_{x_1}(\mathcal{G}(p_1, \alpha(x_1), f, g))(Z_1)$ and $Z \in \mathcal{A}^{p_2}_{x_2}(\mathcal{G}(p_2, \alpha(x_2), f, g))(Z_2)$. From Equation (11.4), it follows that if these memberships hold, then the pair of memberships $Z \in \mathcal{G}(p_1, \alpha(x_1), f, g)(y_1)$ and $Z \in \mathcal{G}(p_2, \alpha(x_2), f, g)(y_2)$ also hold, for some $y_1, y_2 \in Y$. Now, Lemma 11.8 entails that either $x_1 = x_2$, $p_1 = p_2$, $y_1 = y_2$, or $x_1 = x_2$ and $p_1 \neq p_2$. In either case, $Z_1 \overset{X}{\sim} Z_2$. For the former case, we have equality of the states and Y variables, which, in combination with Lemma 11.7, implies $Z_1 = Z_2$. In the latter case, Z_1 and Z_2 are not indexed by the same state, so $Z_1 \not\equiv Z_2$. □

Lemma 11.10

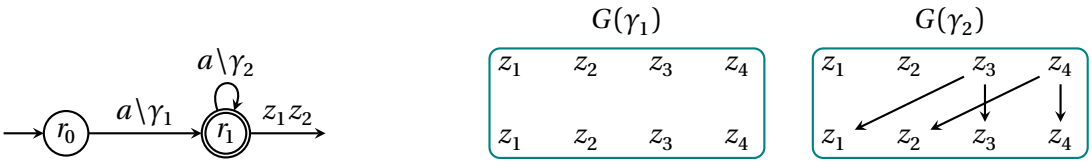
For each assignment $\gamma : \mathcal{Z} \to [\mathcal{Z}, \Sigma_3]$, if there exist variables $Z, Z_1, Z_2 \in \mathcal{Z}$ such that $Z_1 \in \gamma(Z)$ and $Z_2 \in \gamma(Z)$, then $Z_1 \equiv Z_2$ or $Z_1 \overset{X}{\sim} Z_2$.

Proof. If $Z_1, Z_2 \in \gamma(Z)$, then $Z_1, Z_2 \in \mathcal{G}(p, \alpha(x), f, g)(y)$ for some y. Both variables must be introduced by a shape summary constructed by $\mathcal{G}(p, \alpha(x), f, g)$. This yields two cases: (1) both variables belong to a summary mapping $g^{p'}_{x'}$ with common parameters from P and X, or (2) each variable belongs to summary mappings $g^{p_1}_{x_1}$ and $g^{p_2}_{x_2}$ with distinct parameters. When Z_1, Z_2 both occur in some element of $\bigcup_{y \in Y} g(p', x', y)$, then $Z_1 \equiv Z_2$ by Lemma 11.4. Otherwise, when Z_1 occurs in an element of $\bigcup_{y \in Y} g(p_1, x_1, y)$ and Z_2 occurs in some element of $\bigcup_{y \in Y} g(p_2, x_2, y)$, it must hold that $x_1 \neq x_2$ because α is copyless. Since the $\mathcal{Z}$ variables in question cannot be indexed by the same X variable, we conclude that $Z_1 \overset{X}{\sim} Z_2$. □

Theorem 11.3

If $T^1_3 = T^2_3 \circ T^1_2$ is constructed according to Definition 11.6, then T^1_3 is diamond-free.

Proof. Lemma 11.10 and Lemma 11.9 imply that any pair of copies of a single variable never occur in a common variable later in the computation. Therefore, it is impossible for a diamond to occur in any flow graph of a run in T^1_3. □



(a) T_3 (b) Flow graphs of γ_1 and γ_2.

Figure 11.9: The SST T_3 and flow graphs for each of its assignments.

11.3 Eliminating Copyful Assignments From Diamond-Free Transducers

In this section, provide a procedure that produces an equivalent copyless NSST when given a diamond-free NSST. The SST T_3 from Figure 11.2, also displayed in Figure 11.9 along with the flow graphs of its assignments, will be used as a running example to illustrate the argument.

11.3.1 Assignment Decomposition

The first portion of the conversion procedure involves decomposing each copyful assignment α in the given diamond-free NSST into a collection of copyless assignments that "cover" α in the sense that the union of these copyless mappings coincides exactly with the mapping α. This decomposition subroutine, `Decompose`, is given in Algorithm 7. Figure 11.10 provides an illustration of the execution of `Decompose` when given the copyful assignment γ_2 of T_3 as input.

Algorithm 7: Decomposition of a copyful assignment into copyless assignments.

```
1  function Decompose(α)
2      if α is copyless then
3          return {α}
4      Initialize D as ∅;
5      Choose x ∈ X for which there exist distinct y, z ∈ X such that x ∈ α(y) and x ∈ α(z);
6      foreach y ∈ X such that x ∈ α(y) do
7          foreach z ∈ X \ {y} such that x ∈ α(z) do
8              Let β be a new assignment that is identical to α;
9              Set β(z) to ε;
10             if β is copyless then
11                 Set D to D ∪ {β};
12             else
13                 Set D to D ∪ Decompose(β);
14     return D;
```

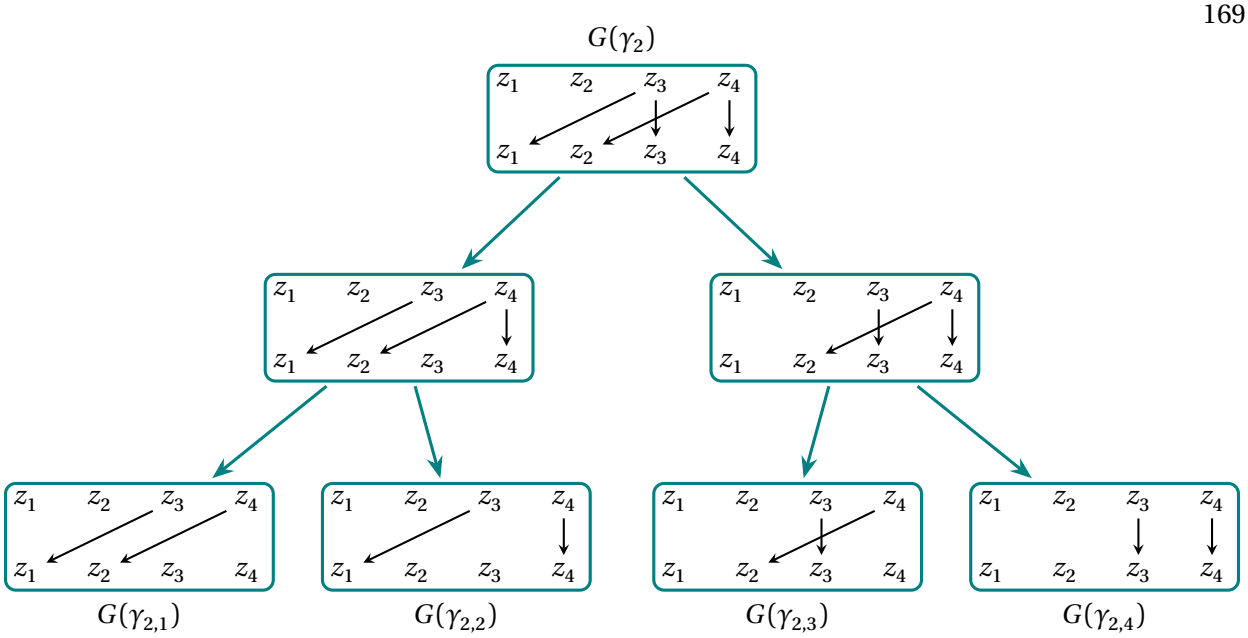


Figure 11.10: Decomposition of the copyful assignment γ_2 of T_3.

Lemma 11.11

If α is diamond-free, then each $\alpha' \in \texttt{Decompose}(\alpha)$ is copyless and

$$\alpha = \bigcup_{\alpha' \in \texttt{Decompose}(\alpha)} \alpha'.$$

Proof. Each α_k being copyless follows from the fact that the conditionals at lines 2 and 10 of Algorithm 7 ensure only copyless assignments are added to D. If the conditional at line 2 is entered then $\{\alpha\}$ is returned and $\alpha = \alpha$, thus satisfying the second assertion of the lemma. Otherwise, a variable x is selected such that x is copied by α. The subsequent loop iterates through the possible target variables of for x, fixes one such y at each iteration, creates a new assignment β as an identical copy of α, and then deletes $\beta(z)$ for all other variables z such that $x \in \alpha(z)$. Thus, every mapping in the assignment α is present in β for at least one iteration of the loop. Furthermore, this process either adds each β into D or recurses with β as a parameter. The former guarantees that at least one assignment in D includes any given mapping from α. In the latter case, a mapping will not be removed later on, since the variable x will no longer be copied by any assignment β passed recursively to `Decompose`. Therefore, every piece of α occurs in some assignment in D and $\alpha = \bigcup_{\alpha' \in \texttt{Decompose}(\alpha)} \alpha'$. □

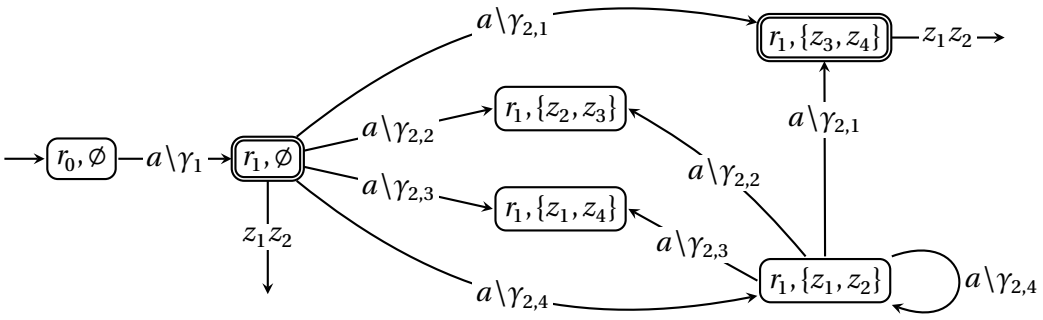


Figure 11.11: A copyless functional NSST equivalent to T_3 from Figure 11.2 and constructed according to Definition 11.7.

11.3.2 From Diamond-Free Transducers To Copyless Transducers

Now that we have established the correctness of the decomposition function, we are prepared to define the construction for transforming a diamond-free NSST into an equivalent copyless NSST.

Definition 11.7 – Diamond-Free → Copyless

Suppose that $T = \langle \Sigma, \Lambda, Q, q_0, X, A, \delta, F \rangle$ is a diamond-free NSST. Let $T' = \langle \Sigma, \Lambda, Q', q_0', X, A', \delta', F' \rangle$ be a copyless NSST where

- $Q' = Q \times 2^X$ and $q_0' = \langle q_0, \emptyset \rangle$;
- $A' = \bigcup_{\alpha \in A} \texttt{Decompose}(\alpha)$;
- $\langle \langle p, Y \rangle, s, \beta, \langle q, Z \rangle \rangle \in \delta'$ if the following conditions all hold:
  (1) no pair $x, y \in X$ exists such that $y \in Y$ and $y \in \beta(x)$;
  (2) there exists $\alpha \in A$ such that $\langle p, s, \alpha, q \rangle \in \delta$;
  (3) $y \in Z$ iff there exists $x \in X$ such that $x \in \alpha(y)$ and $x \notin \beta(y)$;
- $F'(\langle q, Y \rangle) = F(q)$ if $x \notin F(q)$ for all $x \in Y$.

In the sequel, suppose that $T = \langle \Sigma, \Lambda, Q, q_0, X, A, \delta, F \rangle$ is a diamond-free NSST. Additionally suppose that $T' = \langle \Sigma, \Lambda, Q', q_0', X, A', \delta', F' \rangle$ is the copyless NSST constructed according to Definition 11.7.

Lemma 11.12

If ρ is an arbitrary run $q_0 \xrightarrow[\alpha_1]{w_1} \dots \xrightarrow[\alpha_n]{w_n} q_n$ in T and $\mathcal{R}(\rho)$ is the set of all runs of the form $\langle q_0, \emptyset \rangle \xrightarrow[\beta_1]{w_1} \dots \xrightarrow[\beta_n]{w_n} \langle q_n, Y_n \rangle$ in T', then

$$\mathcal{V}_\rho = \bigcup_{\rho' \in \mathcal{R}(\rho)} \mathcal{V}_{\rho'}$$

Proof. We proceed by induction on the run ρ.

Base case: ρ is the empty run.

In this case, $\mathcal{R}(\rho)$ is a singleton set containing the empty run and thus $\mathcal{V}_\rho = \bigcup_{\rho' \in \mathcal{R}(\rho)} \mathcal{V}_{\rho'}$.

Inductive case: $\sigma = \rho \xrightarrow[\alpha]{s} r$ is a run in T on ws and $\mathcal{V}_\rho = \bigcup_{\rho' \in \mathcal{R}(\rho)} \mathcal{V}_{\rho'}$.

The valuation for σ in T may be written as $\mathcal{V}_\sigma = \mathcal{V}_\rho \circ \alpha$, and $\mathcal{R}(\sigma)$ may be written as

$$\mathcal{R}(\sigma) = \bigcup_{\beta \in \texttt{Decompose}(\alpha)} \bigcup_{\rho' \in \mathcal{R}(\rho)} \left\{ \rho' \xrightarrow[\beta]{s} \langle r, Z \rangle : \langle \langle q_n, Y_n \rangle, s, \alpha, \langle r, Z \rangle \rangle \in \delta' \right\}.$$

As a result, we have that

$$\bigcup_{\sigma' \in \mathcal{R}(\sigma)} \mathcal{V}_{\sigma'} = \bigcup_{\beta \in \texttt{Decompose}(\alpha)} \bigcup_{\rho' \in \mathcal{R}(\rho)} \mathcal{V}_{\rho'} \circ \beta.$$

Applying the inductive hypothesis and Lemma 11.11 to this equation, we obtain the desired equivalence:

$$\begin{aligned}
\bigcup_{\sigma' \in \mathcal{R}(\sigma)} \mathcal{V}_{\sigma'} &= \bigcup_{\beta \in \texttt{Decompose}(\alpha)} \bigcup_{\rho' \in \mathcal{R}(\rho)} \mathcal{V}_{\rho'} \circ \beta \\
&= \bigcup_{\beta \in \texttt{Decompose}(\alpha)} \mathcal{V}_\rho \circ \beta \\
&= \mathcal{V}_\rho \circ \bigcup_{\beta \in \texttt{Decompose}(\alpha)} \beta \\
&= \mathcal{V}_\rho \circ \alpha \\
&= \mathcal{V}_\sigma.
\end{aligned}$$

□

Now, we are ready to state the main result of this section.

Theorem 11.4 – Diamond-Free → Copyless

Every diamond-free NSST with n states and m variables is equivalent to a copyless NSST with $n2^m$ states and m variables.

The proof of correctness is carried out in two parts through Lemmas 11.13 and 11.14. The number of states in the resulting copyless NSST is straight forward from Definition 11.7.

Lemma 11.13

If $\langle w, w' \rangle \in [\![T]\!]$, then $\langle w, w' \rangle \in [\![T']\!]$.

Proof. Assuming $\langle w, w' \rangle \in [\![T]\!]$, then there must be a run $\rho = q_0 \xrightarrow[\alpha_1]{w_1} \dots \xrightarrow[\alpha_n]{w_n} q_n$ in T on w such that $\mathcal{V}_\rho \circ F(q_n) = w'$. By Lemma 11.12, it holds that

$$w' = \bigcup_{\rho' \in \mathcal{R}(\rho)} \mathcal{V}_{\rho'} \circ F(q_n),$$

and, by Definition 11.7, it holds, for any $Y_n \subseteq X$, that

$$w' = \bigcup_{\rho' \in \mathcal{R}(\rho)} \mathcal{V}_{\rho'} \circ F'(\langle q_n, Y_n \rangle),$$

such that there is no variable x occurring both in Y_n and in $F(q_n)$. Each run in $\mathcal{R}(\rho)$ corresponds to a nondeterministic choice resolving which copies induced by assignments in ρ will be used later in the computation. Since each assignment in ρ is diamond-free, no two copies can be combined in a later assignment, and thus there at least one run in $\mathcal{R}(\rho)$ corresponding to the appropriate choice of copies for simulating ρ. Therefore, $\langle w, w' \rangle \in [\![T]\!]$ implies $\langle w, w' \rangle \in [\![T']\!]$. □

Lemma 11.14

If $\langle w, w' \rangle \in [\![T']\!]$, then $\langle w, w' \rangle \in [\![T]\!]$.

Proof. Assuming $\langle w, w' \rangle \in [\![T']\!]$, there exists a run $\rho' = \langle q_0, \emptyset \rangle \xrightarrow[\beta_1]{w_1} \dots \xrightarrow[\beta_n]{w_n} \langle q_n, Y_n \rangle$ in T' on w such that

$$w' = \mathcal{V}_{\rho'} \circ F'(\langle q_n, Y_n \rangle).$$

By Definition 11.7, there exists a transition $\langle q_{k-1}, w_k, \alpha_k, q_k \rangle \in \delta$ in T', for each transition

$$\langle \langle q_{k-1}, Y_{k-1} \rangle, w_k, \beta_k, \langle q_k, Y_k \rangle \rangle \in \delta'$$

ocurring in the run ρ' of T, such that the memberships $\beta_k \in \texttt{Decompose}(\alpha_k)$ and $y \in Y_k$ hold iff the memberships $x \in \alpha_k(y)$ and $x \notin \beta_k(y)$ hold for some $x \in X$. Therefore, we may construct the run $\rho = q_0 \xrightarrow[\alpha_1]{w_1} \dots \xrightarrow[\alpha_n]{w_n} q_n$ for which $\rho' \in \mathcal{R}(\rho)$ by stringing together these transitions from δ. This shows that every accepting run ρ' in T' on w has a corresponding accepting run ρ in T on w such that

$$\mathcal{V}_\rho \circ F(q_n) = \mathcal{V}_{\rho'} \circ F'(\langle q_n, Y_n \rangle).$$

Hence, $\langle w, w' \rangle \in [\![T']\!]$ implies $\langle w, w' \rangle \in [\![T]\!]$. □

Part IV

Conclusion & Outlook

Chapter 12

Summary

This thesis has presented a broad investigation into the connection between regular transformations and sequential optimization. The fruits of this inquiry include (1) direct applications of regular transformations towards modeling aspects of optimization problems, (2) original fundamental problems discovered through pushing the bounds of regularity in sequential optimization, and (3) foundational results in the theory of regular transformations, motivated by from the sequential optimization perspective.

Regarding direct applications, we have seen two main developments. The introduction of probabilistic reward machines and the design of the RL* algorithm extend the reach of the reward machine approach for reinforcement with non-Markovian reward signals. Regular Markov decision processes and regular reinforcement learning, on the other hand, provide a new avenue for incorporating aspects of regularity into optimization.

Our investigation into concrete applications also lead to the discovery of new research directions in the theory of optimization. Through attempting to push the reward machine approach further, we introduced a notion of register reward machines that inspired an optimization-oriented treatment of depreciating assets as well as the theoretically unusual past-discounted objectives.

Finally, we have made progress on the theory of regular transductions, with a view towards extending our work on regular Markov decision process. We established and justified a notion of regular relations over infinite strings, designed an extension of the regular mode checking framework to incorporate these relations, and developed theoretical results regarding composition and type checking

transducers that make this framework possible.

In conclusion, the work presented in this dissertation has deepened the connection between regular transformations and sequential optimization, by illuminating the multifaceted nature of this relationship.

Chapter 13

Future Research Prospects

In this final chapter, we discuss some avenues for further development of the research presented in this thesis.

13.1 Register Reward Machines

As mentioned in Chapter 6, there are remaining obstacles that block the path forward for the use of register reward machines in reinforcement learning problems. The main challenge here is that the product construction that is so heavily relied upon in the reward machine literature is not appropriate for register reward machines. New ideas will be required to handle the problem of dealing with the fact that register reward machines generate reward signals with infinitely many individual reward values.

13.2 Generating Functions & Formal Power Series

The mathematics of discounting can be viewed as the culmination of results related to optimization that can be obtained by squeezing as much as possible from the most basic identity around geometric series: $\sum_{n=1}^{\infty} cz^{n-1} = \frac{c}{1-z}$, for $z \in [0,1)$. The same might be said of the theory of *ordinary generating functions* [Wil90], which studies sequences $(c_n)_{n\geq 1}$ by means of associated *formal power series* $\sum_{n=1}^{\infty} c_n z^{n-1}$. For example, the generating function of the constant sequence $(c)_{n\geq 1}$ is the mapping $z \mapsto \frac{c}{1-z}$ and is represented by the formal power series $\sum_{n=1}^{\infty} cz^{n-1}$.

A basic operation on formal power series is the Cauchy product

$$(F * G)(z) = \sum_{n=1}^{\infty} z^{n-1} \sum_{k=1}^{n} c_k d_{n-k}$$

for $F(z) = \sum_{n=1}^{\infty} c_n z^{n-1}$ and $G(z) = \sum_{n=1}^{\infty} d_n z^{n-1}$, and the most fundamental result about Cauchy products is the identity

$$(F * G)(z) = F(z)G(z),$$

which holds whenever z is in the appropriate region of convergence. This equation is precisely a generalized version of Mertens' Theorem, which was the key ingredient to the proof of Theorem 7.1.

This observation illuminates an intriguing connection between generating functions and the theory of depreciating assets developed in the current work. It suggests that the theory of generating functions and formal power series have more to offer in the context of optimization problems over general dynamic (*i.e.* not restricted to only depreciation dynamics) asset streams.

13.3 Computing Past-Discounted Values In General Environments

In Chapter 8, we developed techniques for computing optimal past-discounted values exactly in deterministic MDPs and in ergodic MDPs, but not in general MDPs. Addressing the exact computation of optimal past-discounted values in general (finite) MDPs is another direction for future work. Resolving this open problem requires either the development of an algorithm for computing these values, or a proof that such values are not computable. We conjecture that these values are computable, but the methods used in the special cases mentioned above do not seem to readily generalize. We further conjecture that some adaptation of the approach used for deterministic environments—based on the graph transpose operation—could be designed to account for the stochastic transitions present in general MDPs.

13.4 Temporal Logic

Temporal logic [GRF00; Lam83; Pnu77; RU71] deals with formal logical systems with operators related to time. In a typical setting, these operators allow one to express things like "eventually ϕ" and "at all times ϕ", relative to some property ϕ. Research on temporal logic has substantial intersection with game theory. On one hand, temporal logic has been applied in game theory to specify game

objectives [Boz+21; HRS05; MT02; WT16] and reason about equilibria [Ven04; Ven11]. On the other hand, game theory is applied to temporal logic to provide semantics [GKR17; GKR21; HKR20; HKR22] and prove expressivity results [EW00]. Temporal logics have also found applications in computer science, where they have been widely adopted as property specification languages [BBP89]. While there are many varieties of temporal logic, the remainder of this section primarily focuses on *linear temporal logic* (LTL)[1] .

Discounted Semantics

Notions of time preference based on discounting have been incorporated into temporal logics such as LTL [ABK14; ABK16; Alu+23; Man12], computation tree logic [Alf+05], and the modal μ-calculus [AHM03]. All such work involves (1) extension of a logic's boolean semantics to have quantitative semantics and (2) definition of discounted versions of the typical logical operators that are compatible with the quantitative semantics.

A significant aspect that splits the existing work is whether or not each particular notion discounting is cumulative. Cumulative discounting, like that considered in the current article, is based on sequences $\left(\sum_{k=1}^{n} \lambda^k r_k\right)_{n\in\mathbb{N}}$ of partial sums, while non-cumulative discounting is based on sequences $(\lambda^n r_n)_{n\in\mathbb{N}}$ of individual scaled terms. The works of Almagor, Boker, and Kupferman [ABK14; ABK16] and Alur et al. [Alu+23] use non-cumulative discounted semantics for their logics. In contrast, cumulative discounted semantics are used in the work of Mandrali [Man12].

Past-Tense Modalities

Within the algorithmically focussed research around temporal logic, there is a strong future-oriented bias. The vast majority of this body of work considers logics that are exclusively future tense, *i.e.* that do not include any temporal operators which reference the past. Despite their relative unpopularity, logics incorporating past tense operators have also been considered [Gab87; Lan05; LS94; LS95; LPZ85; Mar02; Mar03; Mar04]. The logic that extends LTL by including past tense operators in addition to all of

[1] Other notable types include branching-time temporal logic [BPM83] and alternating-time temporal logic [AHK97; AHK02].

the usual future tense operators will be referred to as pLTL in the remaining discussion.

Perhaps surprisingly, Lichtenstein, Pnueli, and Zuck [LPZ85] show that adding past tense operators to LTL does not change the expressive power of the logic: a property is definable in pLTL if, and only if, it is definable in LTL. A notable result of Markey [Mar03] complements this, stating that despite the two logics having equivalent expressive power, pLTL is exponentially more succinct than LTL: anything that can be stated by a pLTL formula can be stated by an LTL formula, but the smallest equivalent LTL formula may be exponentially larger than the original pLTL formula.

While semantics of classical LTL are defined with respect to infinite traces, there has been recent interest, especially in the artificial intelligence community [BGP18; BG19a; GV15; Gia+19], in variations having semantics defined with respect to finite traces. The logic LTL_f [GV13], for instance, is the analog of LTL for finite traces. The logic pLTL_f, which extends LTL_f by adding past tense operators, and its derivative *pure* pLTL_f or ppLTL_f [Gia+20], in which only past tense operators are allowed, have been studied as well. Recent work of Artale et al. [Art+23] shows that LTL_f and ppLTL_f, despite being equivalently expressive, are incomparable in terms of succinctness: there exist formulae in of each of these logics such that the smallest equivalent formula in the other is exponentially larger.

Past-Discounting In Temporal Logic

As far as we are aware, the only existing work combining discounted semantics and past tense operators is that of Almagor, Boker, and Kupferman [ABK16], in which a discounted variant of LTL is their main topic of study and discounted past-tense operators are introduced as an extension to this. They obtain complexity bounds for model-checking of Kripke structures and finite state transducers against specifications written in their version of discounted LTL, and then show that the bounds are unchanged when specifications are taken from the extended logic with discounted past tense operators. The authors mention that it remains open whether adding these discounted past operators adds any expressive power to their discounted LTL, and they conjecture the affirmative. We reiterate that the discounted LTL of Almagor, Boker, and Kupferman [ABK16] is non-cumulative, making it difficult to see how the present work could be applied to their setting.

On the other hand, an exciting avenue for future work is to investigate how the ideas of from Chapters 7 and 8 might be applied towards studying LTL with discounted past operators under cumulative discounted semantics, like those for the logic of Mandrali [Man12]. What can be said regarding the expressive powers of LTL, pLTL, and ppLTL under cumulative discounted semantics? In the event that two of these logics align on expressivity, what can be said about their relative succinctness? Do semantics based on finite-traces change the nature of any such relationships?

13.5 Transition-Regular Markov Decision Processes

In Part III, we made progress towards enabling the extension of the regular Markov decision processes from Chapter 5 to transition-regular Markov decision processes. Despite these advancements, a crucial issue remains that obstructs the extension of the regular reinforcement learning approach in this setting. A key idea in deep regular reinforcement learning is the use graph neural networks to approximate policies over regular Markov decision processes. This is possible due to the fact that every state in an RMDP corresponds to a regular language, and any regular language can be represented by a finite automaton (which amounts essentially to a labeled directed graph). In Chapter 10, it became clear that our graph neural network approach cannot work in the setting of transition-regular decision processes. The reason for this is that regular languages are not closed under regular transduction, as the image of regular transduction can be as complex as a context-sensitive language. Non-regular languages cannot be represented by finite automata, leaving us without a means of applying graph neural networks in the same way. Is there an alternative route that can enable deep regular reinforcement learning over transition-regular MDPs?

To further motivate transition-regular decision processes, consider the difference in expressive power between rational and regular transductions. Since iterated rational transductions are already Turing-complete, there is nothing to be gained from the extension to regular transductions from a computability perspective. On the other hand, there is much to be gained regarding efficient learnability. As an illustrative example, consider the regular reinforcement learning problem of learning to reverse all of the strings in the input language. It is possible to model this problem as an RMDP by including an

action that corresponds to a rational transduction

$$w \mapsto \begin{cases} w'\#\sigma & \text{if } w = w'\sigma \in (\Sigma \smallsetminus \{\#\})^+, \\ w_1\#\sigma\, w_2 & \text{if } w = w_1\#w_2\sigma, \text{ for some } w_1, w_2 \in (\Sigma \smallsetminus \{\#\})^+ \text{ and } \sigma \in \Sigma \smallsetminus \{\#\}, \sigma\, w'\#w, \\ w' & \text{if } w = \#w' \text{ for some } w' \in (\Sigma \smallsetminus \{\#\})^+. \end{cases}$$

This transduction takes each word $w\sigma$ in the input language to $w\#\sigma$, where # is a marker symbol, and then each successive application moves the last symbol before the # to the last position after the #. After $n+1$ applications, this reverses a string of length n:

$$\begin{aligned} & w = \sigma_1 ... \sigma_n \\ & \quad \mapsto \\ & \quad\quad \sigma_1 ... \sigma_{n-1} \# \sigma_n \\ & \quad\quad\quad \mapsto \\ & \quad\quad\quad\quad \sigma_1 ... \sigma_{n-2} \# \sigma_n \sigma_{n-1} \\ & \quad\quad\quad\quad\quad \mapsto ... \mapsto \\ & \quad\quad\quad\quad\quad\quad \sigma_1 \# \sigma_n ... \sigma_2 \\ & \quad\quad\quad\quad\quad\quad\quad \mapsto \\ & \quad\quad\quad\quad\quad\quad\quad\quad \# \sigma_n ... \sigma_1 \\ & \quad\quad\quad\quad\quad\quad\quad\quad\quad \mapsto \\ & \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \sigma_n ... \sigma_1 = \overleftarrow{w}. \end{aligned}$$

Thus, an agent could learn a policy to reverse strings by learning to repeatedly apply this transduction, but if the initial language is infinite the reversed language can never be fully realized in finite time.

By using regular transformations, we can introduce an action that simply corresponds to the regular transduction $w \mapsto \overleftarrow{w}$, allowing the agent to learn a policy that also leads to the realization of the intended outcome of reversing all the strings in the input language. In this fashion, transition-regular decision processes are able to take "big steps", relative to the steps taken by RMDPs. If the technical

challenges can be overcome, this would result in an extremely powerful framework for symbolic RL.

13.6 Verification of Neural Policies

A significant concern related to deep reinforcement learning is the lack of theoretical guarantees that tabular methods enjoy. While there are relatively strong convergence results for tabular RL, deep RL techniques are not known to have provable converge to optimal policies. As a result, analysis and verification of learned neural polices is a crucial part of the deep RL workflow, especially in safety-critical situations.

One approach to verification of neural networks is based on *abstraction*. This involves defining an *abstract domain* and approximating the images of neuron layers with elements of the abstract domain. These abstract domains are often geometric in nature, and their elements are defined as affine images of regions in $\mathbb{R}^n$. By iteratively approximating outputs of neuron layers, overapproximations of the entire network's image can be obtained and verified against properties related to safety and robustness. See Appendix A[2] for a more detailed account of this framework that includes further research not presented in the main body of this thesis.

Through the abstraction based approach to neural network verification, we find a new avenue for investigating the connection between optimization and regularity. As demonstrated by the modified tangrams case study in Chapter 5, it is possible to represent geometric regions as regular languages and affine transformations as rational transductions. Using similar ideas, one could use regular languages as an abstract domain in this verification context. This would facilitate the application of techniques from regular model checking to verification of neural policies learned through deep RL algorithms. What advantages and disadvantages would such an abstract domain have over traditional geometric domains? What do we get from having regularity baked into an abstract domain, which is both geometric and symbolic?

[2] Appendix A is based on an FM 2023 paper [Bak+23].

Bibliography

[AB20] Eden Abadi and Ronen I. Brafman. “Learning and Solving Regular Decision Processes”. In: *International Joint Conference on Artificial Intelligence, IJCAI*. ijcai.org, 2020, pp. 1948–1954. DOI: 10.24963/ijcai.2020/270.

[Abd+02] Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d’Orso. “Regular Model Checking Made Simple and Efficient”. In: *International Conference on Concurrency Theory, CONCUR*. Vol. 2421. Springer, 2002, pp. 116–130. DOI: 10.1007/3-540-45694-5_9.

[Abd+04] Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Mayank Saksena. “A Survey of Regular Model Checking”. In: *International Conference on Concurrency Theory, CONCUR*. Vol. 3170. Springer, 2004, pp. 35–48. DOI: 10.1007/978-3-540-28644-8_3.

[Abd+12] Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, Julien d’Orso, and Mayank Saksena. “Regular model checking for LTL(MSO)”. In: *International Journal on Software Tools for Technology Transfer* 14.2 (2012), pp. 223–241. DOI: 10.1007/s10009-011-0212-z.

[ABK14] Shaull Almagor, Udi Boker, and Orna Kupferman. “Discounting in LTL”. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS*. Vol. 8413. Springer, 2014, pp. 424–439. DOI: 10.1007/978-3-642-54862-8_37.

[ABK16] Shaull Almagor, Udi Boker, and Orna Kupferman. “Formally Reasoning About Quality”. In: *Journal of the ACM* 63.3 (2016), 24:1–24:56. DOI: 10.1145/2875421.

[ABZ92] A. L. Ambler, M. M. Burnett, and B. A. Zimmerman. “Operational versus definitional: a perspective on programming paradigms”. In: *Computer* 25.9 (Sept. 1992), pp. 28–43. DOI: 10.1109/2.156380.

[AČ10] Rajeev Alur and Pavol Černý. “Expressiveness of streaming string transducers”. In: *Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010. DOI: 10.4230/LIPIcs.FSTTCS.2010.1.

[AČ11] Rajeev Alur and Pavol Černý. “Streaming transducers for algorithmic verification of single-pass list-processing programs”. In: *Symposium on Principles of Programming Languages, POPL*. ACM, 2011. DOI: 10.1145/1926385.1926454.

[Ack73] Susan Rose Ackerman. “Used cars as a depreciating asset”. In: *Economic Inquiry* 11.4 (1973), p. 463. DOI: 10.1111/j.1465-7295.1973.tb00975.x.

[ACM12] Konstantin Avrachenkov, Laura Cottatellucci, and Lorenzo Maggi. “Algorithms for uniform optimal strategies in two-player zero-sum stochastic games with perfect information”. In: *Oper. Res. Lett.* 40.1 (2012), pp. 56–60. DOI: 10.1016/j.orl.2011.10.005.

[AD11] Rajeev Alur and Jyotirmoy V. Deshmukh. “Nondeterministic Streaming String Transducers”. In: *International Colloquium on Automata, Languages and Programming, ICALP*. Springer, 2011. DOI: 10.1007/978-3-642-22012-8_1.

[AD12] Rajeev Alur and Loris D’Antoni. “Streaming Tree Transducers”. In: *International Colloquium on Automata, Languages, and Programming, ICALP*. Vol. 7392. Springer, 2012, pp. 42–53. DOI: 10.1007/978-3-642-31585-5_8.

[AD17] Rajeev Alur and Loris D’Antoni. “Streaming Tree Transducers”. In: *Journal of the ACM* 64.5 (2017), 31:1–31:55. DOI: 10.1145/3092842.

[AFR14] Rajeev Alur, Adam Freilich, and Mukund Raghothaman. "Regular combinators for string transformations". In: *Joint Meeting of the Conference on Computer Science Logic (CSL) and the Symposium on Logic in Computer Science (LICS), CSL-LICS*. ACM, 2014, 9:1–9:10. DOI: 10.1145/2603088.2603151.

[AFT12] Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. "Regular Transformations of Infinite Strings". In: *Symposium on Logic in Computer Science, LICS*. IEEE, 2012. DOI: 10.1109/LICS.2012.18.

[AHK02] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. "Alternating-time temporal logic". In: *Journal of the ACM* 49.5 (2002), pp. 672–713. DOI: 10.1145/585265.585270.

[AHK97] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. "Alternating-time Temporal Logic". In: *Symposium on Foundations of Computer Science, FOCS*. IEEE, 1997, pp. 100–109. DOI: 10.1109/SFCS.1997.646098.

[AHM03] Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. "Discounting the Future in Systems Theory". In: *International Colloquium on Automata, Languages and Programming, ICALP*. Vol. 2719. Springer, 2003, pp. 1022–1037. DOI: 10.1007/3-540-45061-0_79.

[Aki+18] Michael Akintunde, Alessio Lomuscio, Lalit Maganti, and Edoardo Pirovano. "Reachability Analysis for Neural Agent-Environment Systems". In: *International Conference on Principles of Knowledge Representation and Reasoning, KR*. AAAI Press, 2018, pp. 184–193. URL: https://aaai.org/ocs/index.php/KR/KR18/paper/view/17991.

[Aks+13] S. Akshay, Nathalie Bertrand, Serge Haddad, and Loïc Hélouët. "The Steady-State Control Problem for Markov Decision Processes". In: *International Conference on Quantitative Evaluation of Systems, QEST*. Vol. 8054. Springer, 2013, pp. 290–304. DOI: 10.1007/978-3-642-40196-1_26.

[Alb21] Aws Albarghouthi. *Introduction to Neural Network Verification*. verifieddeeplearning.com, 2021. DOI: 10.48550/arXiv.2109.10317. arXiv: 2109.10317 [cs.LG]. URL: http://verifieddeeplearning.com.

[Alf+05] Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. "Model checking discounted temporal properties". In: *Theoretical Computer Science* 345.1 (2005), pp. 139–170. DOI: 10.1016/j.tcs.2005.07.033.

[Alu+05] Rajeev Alur, Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan. "Congruences for Visibly Pushdown Languages". In: *International Colloquium on Automata, Languages and Programming, ICALP*. Vol. 3580. Springer, 2005, pp. 1102–1114. DOI: 10.1007/11523468_89.

[Alu+13] Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. "Regular Functions and Cost Register Automata". In: *Symposium on Logic in Computer Science, LICS*. IEEE, 2013, pp. 13–22. DOI: 10.1109/LICS.2013.65.

[Alu+23] Rajeev Alur, Osbert Bastani, Kishor Jothimurugan, Mateo Perez, Fabio Somenzi, and Ashutosh Trivedi. "Policy Synthesis and Reinforcement Learning for Discounted LTL". In: *International Conference on Computer Aided Verification, CAV*. Vol. 13964. Springer, 2023, pp. 415–435. DOI: 10.1007/978-3-031-37706-8_21.

[AM04] Rajeev Alur and P. Madhusudan. "Visibly pushdown languages". In: *Symposium on Theory of Computing, STOC*. ACM, 2004, pp. 202–211. DOI: 10.1145/1007352.1007390.

[AM09] Daniel Andersson and Peter Bro Miltersen. “The Complexity of Solving Stochastic Games on Graphs”. In: *International Symposium on Algorithms and Computation, ISAAC*. Vol. 5878. Springer, 2009, pp. 112–121. DOI: 10.1007/978-3-642-10631-6_13.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993. ISBN: 978-0-13-617549-0.

[Ang87] Dana Angluin. “Learning Regular Sets from Queries and Counterexamples”. In: *Information and Computation* 75.2 (1987), pp. 87–106. DOI: 10.1016/0890-5401(87)90052-6.

[AR13] Rajeev Alur and Mukund Raghothaman. “Decision Problems for Additive Regular Functions”. In: *International Colloquium on Automata, Languages, and Programming, ICALP*. Vol. 7966. Springer, 2013, pp. 37–48. DOI: 10.1007/978-3-642-39212-2_7.

[Arn85] André Arnold. “A Syntactic Congruence for Rational omega-Language”. In: *Theoretical Computer Science* 39 (1985), pp. 333–335. DOI: 10.1016/0304-3975(85)90148-3.

[Art+23] Alessandro Artale, Luca Geatti, Nicola Gigante, Andrea Mazzullo, and Angelo Montanari. “LTL over Finite Words Can Be Exponentially More Succinct Than Pure-Past LTL, and vice versa”. In: *International Symposium on Temporal Representation and Reasoning, TIME*. Vol. 278. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 2:1–2:14. DOI: 10.4230/LIPIcs.TIME.2023.2.

[Bak+20] Stanley Bak, Hoang-Dung Tran, Kerianne Hobbs, and Taylor T. Johnson. “Improved Geometric Path Enumeration for Verifying ReLU Neural Networks”. In: *International Conference on Computer Aided Verification, CAV*. Vol. 12224. Springer, 2020, pp. 66–96. DOI: 10.1007/978-3-030-53288-8_4.

[Bak+23] Stanley Bak, Taylor Dohmen, K. Subramani, Ashutosh Trivedi, Alvaro Velasquez, and Piotr Wojciechowski. “The Octatope Abstract Domain for Verification of Neural Networks”. In: *International Symposium on Formal Methods, FM*. Vol. 14000. Springer, 2023, pp. 454–472. DOI: 10.1007/978-3-031-27481-7_26.

[Bak21] Stanley Bak. “nnenum: Verification of ReLU Neural Networks with Optimized Abstraction Refinement”. In: *NASA International Symposium Formal Methods, NFM*. Vol. 12673. Springer, 2021, pp. 19–36. DOI: 10.1007/978-3-030-76384-8_2.

[BBP89] Behnam Banieqbal, Howard Barringer, and Amir Pnueli, eds. *Temporal Logic in Specification*. Vol. 398. Springer, 1989. ISBN: 3-540-51803-7. DOI: 10.1007/3-540-51803-7.

[Beh+06] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. “UPPAAL 4.0”. In: *Third International Conference on the Quantitative Evaluation of Systems, QEST*. IEEE, 2006, pp. 125–126. DOI: 10.1109/QEST.2006.59.

[Ber79] Jean Berstel. *Transductions and context-free languages*. Vol. 38. Teubner Studienbücher : Informatik. Teubner, 1979. ISBN: 3519023407. URL: https://www.worldcat.org/oclc/06364613.

[BG19a] Ronen I. Brafman and Giuseppe De Giacomo. “Planning for LTLf /LDLf Goals in Non-Markovian Fully Observable Nondeterministic Domains”. In: *International Joint Conference on Artificial Intelligence, IJCAI*. ijcai.org, 2019, pp. 1602–1608. DOI: 10.24963/ijcai.2019/222.

[BG19b] Ronen I. Brafman and Giuseppe De Giacomo. “Regular Decision Processes: A Model for Non-Markovian Domains”. In: *International Joint Conference on Artificial Intelligence, IJCAI*. ijcai.org, 2019, pp. 5516–5522. DOI: 10.24963/ijcai.2019/766.

[BG19c] Ronen I. Brafman and Giuseppe De Giacomo. “Regular Decision Processes: Modelling Dynamic Systems without Using Hidden Variables”. In: *International Conference on Autonomous Agents and MultiAgent Systems, AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 1844–1846. DOI: 10.5555/3306127.3331938.

[BGP18] Ronen I. Brafman, Giuseppe De Giacomo, and Fabio Patrizi. “LTLf/LDLf Non-Markovian Rewards”. In: *Conference on Artificial Intelligence, AAAI*. AAAI Press, 2018, pp. 1771–1778. DOI: 10.1609/aaai.v32i1.11572.

[BGR19] Benjamin Bordais, Shibashis Guha, and Jean-François Raskin. “Expected Window Mean-Payoff”. In: *Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*. Vol. 150. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 32:1–32:15. DOI: 10.4230/LIPIcs.FSTTCS.2019.32.

[BH77] Meera Blattner and Tom Head. “Single-valued a-transducers”. In: *Journal of Computer and System Sciences* 15.3 (Dec. 1977), pp. 310–327. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(77)80033-0.

[BHV04] Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. “Abstract Regular Model Checking”. In: *International Conference on Computer Aided Verification, CAV*. Vol. 3114. Springer, 2004, pp. 372–386. DOI: 10.1007/978-3-540-27813-9_29.

[BJS09] Mokhtar S Bazaraa, John J Jarvis, and Hanis D Sherali. “Minimal–cost Network Flows”. In: *Linear Programming and Network Flows*. John Wiley & Sons, Ltd, 2009. Chap. 9, pp. 453–512. ISBN: 9780471703778. DOI: 10.1002/9780471703778.ch9.

[BJW05] Bernard Boigelot, Sébastien Jodogne, and Pierre Wolper. “An effective decision procedure for linear arithmetic over the integers and reals”. In: *ACM Transactions on Computational Logic* 6.3 (2005), pp. 614–633. DOI: 10.1145/1071596.1071601.

[BK76] Truman Bewley and Elon Kohlberg. “The Asymptotic Theory of Stochastic Games”. In: *Mathematics of Operations Research* 1.3 (1976), pp. 197–208. DOI: 10.1287/moor.1.3.197.

[BK78] Truman Bewley and Elon Kohlberg. “On Stochastic Games with Stationary Optimal Strategies”. In: *Mathematics of Operations Research* 3.2 (1978), pp. 104–125. DOI: 10.1287/moor.3.2.104.

[BLJ21] Stanley Bak, Changliu Liu, and Taylor T. Johnson. “The Second International Verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results”. In: *CoRR* abs/2109.00498 (2021). DOI: 10.48550/arXiv.2109.00498. arXiv: 2109.00498.

[BLW03] Bernard Boigelot, Axel Legay, and Pierre Wolper. “Iterating Transducers in the Large (Extended Abstract)”. In: *International Conference on Computer Aided Verification, CAV*. Vol. 2725. Springer, 2003, pp. 223–235. DOI: 10.1007/978-3-540-45069-6_24.

[BLW04a] Bernard Boigelot, Axel Legay, and Pierre Wolper. “Omega-Regular Model Checking”. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS*. Vol. 2988. Springer, 2004, pp. 561–575. DOI: 10.1007/978-3-540-24730-2_41.

[BLW04b] Ahmed Bouajjani, Axel Legay, and Pierre Wolper. “Handling Liveness Properties in (*omega*-)Regular Model Checking”. In: *International Workshop on Verification of Infinite-State Systems, INFINITY*. Vol. 138. Elsevier, 2004, pp. 101–115. DOI: 10.1016/j.entcs.2005.02.061.

[BN23] Mikolaj Bojanczyk and Lê Thành Dung Nguyên. “Algebraic Recognition of Regular Functions”. In: *International Colloquium on Automata, Languages, and Programming, ICALP*. Vol. 261. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 117:1–117:19. DOI: 10.4230/LIPIcs.ICALP.2023.117.

[Bok18] Udi Boker. “Why These Automata Types?” In: *International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR*. Vol. 57. EasyChair, 2018, pp. 143–163. DOI: 10.29007/c3bj.

[Bor+11] Endre Boros, Khaled M. Elbassioni, Mahmoud Fouz, Vladimir Gurvich, Kazuhisa Makino, and Bodo Manthey. “Stochastic Mean Payoff Games: Smoothed Analysis and Approximation Schemes”. In: *International Colloquium on Automata, Languages and Programming, ICALP*. Vol. 6755. Springer, 2011, pp. 147–158. DOI: 10.1007/978-3-642-22006-7_13.

[Bor+13] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. “On discounted approximations of undiscounted stochastic games and Markov decision processes with limited randomness”. In: *Oper. Res. Lett.* 41.4 (2013), pp. 357–362. DOI: 10.1016/j.orl.2013.04.006.

[Bor+17] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. “A convex programming-based algorithm for mean payoff stochastic games with perfect information”. In: *Optimization Letters* 11.8 (2017), pp. 1499–1512. DOI: 10.1007/s11590-017-1140-y.

[Bor+18] Endre Boros, Khaled M. Elbassioni, Mahmoud Fouz, Vladimir Gurvich, Kazuhisa Makino, and Bodo Manthey. “Approximation Schemes for Stochastic Mean Payoff Games with Perfect Information and Few Random Positions”. In: *Algorithmica* 80.11 (2018), pp. 3132–3157. DOI: 10.1007/s00453-017-0372-7.

[Bor+19] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. “A pseudo-polynomial algorithm for mean payoff stochastic games with perfect information and few random positions”. In: *Information and Computation* 267 (2019), pp. 74–95. DOI: 10.1016/j.ic.2019.03.005.

[Bou+00] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. “Regular Model Checking”. In: *International Conference on Computer Aided Verification, CAV*. Vol. 1855. Springer, 2000, pp. 403–418. DOI: 10.1007/10722167_31.

[Boz+21] Alper Kamil Bozkurt, Yu Wang, Michael M. Zavlanos, and Miroslav Pajic. “Model-Free Reinforcement Learning for Stochastic Games with Linear Temporal Logic Objectives”. In: *International Conference on Robotics and Automation, ICRA*. IEEE, 2021, pp. 10649–10655. DOI: 10.1109/ICRA48506.2021.9561989.

[BPM83] Mordechai Ben-Ari, Amir Pnueli, and Zohar Manna. “The Temporal Logic of Branching Time”. In: *Acta Informatica* 20 (1983), pp. 207–226. DOI: 10.1007/BF01257083.

[Büc60] J. Richard Büchi. “Weak second-order arithmetic and finite automata”. In: *Mathematical Logic Quarterly* 6.1-6 (1960), pp. 66–92. DOI: 10.1002/malq.19600060105.

[Bur72] Oscar R Burt. “A Unified Theory of Depreciation”. In: *Journal of Accounting Research* (1972), pp. 28–57. DOI: 10.2307/2490217.

[BW02] Bernard Boigelot and Pierre Wolper. “Representing Arithmetic Constraints with Finite Automata: An Overview”. In: *International Conference on Logic Programming, ICLP*. Vol. 2401. Springer, 2002, pp. 1–19. DOI: 10.1007/3-540-45619-8_1.

[Cam+19] Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. “LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning”. In: *International Joint Conference on Artificial Intelligence, IJCAI*. ijcai.org, 2019, pp. 6065–6073. DOI: 10.24963/ijcai.2019/840.

[CC77] Patrick Cousot and Radhia Cousot. “Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”. In: *Symposium on Principles of Programming Languages, POPL*. ACM, 1977, pp. 238–252. DOI: 10.1145/512950.512973.

[CDH09] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. “A Survey of Stochastic Games with Limsup and Liminf Objectives”. In: *Internatilonal Colloquium on Automata, Languages and Programming, ICALP*. Vol. 5556. Springer, 2009, pp. 1–15. DOI: 10.1007/978-3-642-02930-1_1.

[CDS11] Krishnendu Chatterjee, Laurent Doyen, and Rohit Singh. “On Memoryless Quantitative Objectives”. In: *Fundamentals of Computation Theory, FCT*. Vol. 6914. Springer, 2011, pp. 148–159. DOI: 10.1007/978-3-642-22953-4_13.

[CE12] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*. Vol. 138. Encyclopedia of mathematics and its applications. Cambridge University Press, 2012. ISBN: 978-0-521-89833-1. URL: http://www.cambridge.org/fr/knowledge/isbn/item5758776/?site%5C_locale=fr%5C_FR.

[CEW58] Irving M. Copi, Calvin C. Elgot, and Jesse B. Wright. “Realization of Events by Logical Nets”. In: *Journal of the ACM* 5.2 (1958), pp. 181–196. DOI: 10.1145/320924.320931.

[CGN22] Jan Corazza, Ivan Gavran, and Daniel Neider. “Reinforcement Learning with Stochastic Reward Machines”. In: *Conference on Artificial Intelligence, AAAI*. AAAI Press, 2022, pp. 6429–6436. DOI: 10.1609/aaai.v36i6.20594.

[CH07] Krishnendu Chatterjee and Thomas A. Henzinger. “Probabilistic Systems with LimSup and LimInf Objectives”. In: *Infinity in Logic and Computation, International Conference, ILC*. Vol. 5489. Springer, 2007, pp. 32–45. DOI: 10.1007/978-3-642-03092-5_4.

[CH08] Krishnendu Chatterjee and Thomas A. Henzinger. “Value Iteration”. In: *25 Years of Model Checking - History, Achievements, Perspectives*. Vol. 5000. Springer, 2008, pp. 107–138. DOI: 10.1007/978-3-540-69850-0_7.

[Cha+13] Krishnendu Chatterjee, Laurent Doyen, Mickael Randour, and Jean-François Raskin. “Looking at Mean-Payoff and Total-Payoff through Windows”. In: *International Symposium on Automated Technology for Verification and Analysis, ATVA*. Vol. 8172. Springer, 2013, pp. 118–132. DOI: 10.1007/978-3-319-02444-8_10.

[Cha+15] Krishnendu Chatterjee, Laurent Doyen, Mickael Randour, and Jean-François Raskin. “Looking at mean-payoff and total-payoff through windows”. In: *Information and Computation* 242 (2015), pp. 25–52. DOI: 10.1016/j.ic.2015.03.010.

[Chi+13] Wai-Ki Ching, Ximin Huang, Michael K. Ng, and Tak-Kuen Siu. *Markov Chains*. Springer, 2013. ISBN: 978-1-4614-6312-2. DOI: 10.1007/978-1-4614-6312-2.

[Cho59] Noam Chomsky. “On Certain Formal Properties of Grammars”. In: *Information and Control* 2.2 (1959), pp. 137–167. DOI: 10.1016/S0019-9958(59)90362-6.

[CI14] Krishnendu Chatterjee and Rasmus Ibsen-Jensen. “The Complexity of Ergodic Mean-payoff Games”. In: *International Colloquium on Automata, Languages, and Programming, ICALP*. Vol. 8573. Springer, 2014, pp. 122–133. DOI: 10.1007/978-3-662-43951-7_11.

[CJ77] Michal Chytil and Vojtech Jákl. “Serial Composition of 2-Way Finite-State Transducers and Simple Programs on Strings”. In: *International Colloquium on Automata, Languages and Programming*. Vol. 52. Springer, 1977, pp. 135–147. DOI: 10.1007/3-540-08342-1_11.

[CKS81] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. “Alternation”. In: *Journal of the ACM* 28.1 (1981), pp. 114–133. DOI: 10.1145/322234.322243.

[CLS19] Michael B. Cohen, Yin Tat Lee, and Zhao Song. “Solving linear programs in the current matrix multiplication time”. In: *Symposium on Theory of Computing, STOC*. ACM, 2019, pp. 938–942. DOI: 10.1145/3313276.3316303.

[CLS21] Michael B. Cohen, Yin Tat Lee, and Zhao Song. “Solving Linear Programs in the Current Matrix Multiplication Time”. In: *Journal of the ACM* 68.1 (2021), 3:1–3:39. DOI: 10.1145/3424305.

[CM12] Krishnendu Chatterjee and Rupak Majumdar. “Discounting and Averaging in Games across Time scales”. In: *International Journal of Foundations of Computer Science* 23.3 (2012), pp. 609–625. DOI: 10.1142/S0129054112400308.

[CM58] Noam Chomsky and George A. Miller. “Finite State Languages”. In: *Information adn Control* 1.2 (1958), pp. 91–112. DOI: 10.1016/S0019-9958(58)90082-2.

[CMH08] Krishnendu Chatterjee, Rupak Majumdar, and Thomas A. Henzinger. “Stochastic limit-average games are in EXPTIME”. In: *International Journal of Game Theory* 37.2 (2008), pp. 219–234. DOI: 10.1007/s00182-007-0110-5.

[CMM78] Stefano Crespi-Reghizzi, Dino Mandrioli, and David F. Martin. “Algebraic Properties of Operator Precedence Languages”. In: *Information and Control* 37.2 (1978), pp. 115–133. DOI: 10.1016/S0019-9958(78)90474-6.

[Cor+09a] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. “Difference constraints and shortest paths”. In: *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. Chap. 24.4, pp. 664–683. ISBN: 978-0-262-03384-8. URL: http://mitpress.mit.edu/books/introduction-algorithms.

[Cor+09b] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. “Shortest paths”. In: *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. Chap. 24, pp. 664–683. ISBN: 978-0-262-03384-8. URL: http://mitpress.mit.edu/books/introduction-algorithms.

[Cou92] Bruno Courcelle. “Monadic Second-Order Graph Transductions”. In: *Colloquium on Trees in Algebra and Programming, CAAP*. Vol. 581. Springer, 1992, pp. 124–144. DOI: 10.1007/3-540-55251-0_7.

[Cou94] Bruno Courcelle. “Monadic Second-Order Definable Graph Transductions: A Survey”. In: *Theoretical Computer Science* 126.1 (1994), pp. 53–75. DOI: 10.1016/0304-3975(94)90268-2.

[CS59] Noam Chomsky and Marcel P. Schützenberger. “The Algebraic Theory of Context-Free Languages”. In: *Computer Programming and Formal Systems*. Vol. 26. Studies in Logic and the Foundations of Mathematics. Elsevier, 1959, pp. 118–161. DOI: 10.1016/S0049-237X(09)70104-1.

[CS76] Ashok K. Chandra and Larry J. Stockmeyer. “Alternation”. In: *Symposium on Foundations of Computer Science*. IEEE, 1976, pp. 98–108. DOI: 10.1109/SFCS.1976.4.

[Cul79] Karel Culik. “Some decidability results about regular and pushdown translations”. In: *Information Processing Letters* 8.1 (Jan. 1979), pp. 5–8. ISSN: 0020-0190. DOI: 10.1016/0020-0190(79)90080-2.

[Dav+21a] Vrunda Dave, Taylor Dohmen, Shankara Narayana Krishna, and Ashutosh Trivedi. *Regular Model Checking with Regular Relations*. 2021. DOI: 10.48550/arXiv.1910.09072. arXiv: 1910.09072 [cs.LO].

[Dav+21b] Vrunda Dave, Taylor Dohmen, Shankara Narayanan Krishna, and Ashutosh Trivedi. “Regular Model Checking with Regular Relations”. In: *International Symposium on Fundamentals of Computation Theory, FCT*. Vol. 12867. Springer, 2021, pp. 190–203. DOI: 10.1007/978-3-030-86593-1_13.

[DGK18] Vrunda Dave, Paul Gastin, and Shankara Narayanan Krishna. “Regular Transducer Expressions for Regular Transformations”. In: *Symposium on Logic in Computer Science, LICS*. ACM, 2018, pp. 315–324. DOI: 10.1145/3209108.3209182.

[DGK22] Vrunda Dave, Paul Gastin, and Shankara Narayanan Krishna. “Regular transducer expressions for regular transformations”. In: *Information and Computation* 282 (2022), p. 104655. DOI: 10.1016/j.ic.2020.104655.

[Dij61] Edsger Wybe Dijkstra. “Algol 60 translation: An Algol 60 translator for the x1 and Making a translator for Algol 60”. In: *Stichting Mathematisch Centrum. Rekenafdeling* (1961).

[DKR82] A. Demers, C. Keleman, and B. Reusch. “On some decidable properties of finite state translations”. In: *Acta Informatica* 17.3 (Aug. 1982), pp. 349–364. ISSN: 1432-0525. DOI: 10.1007/BF00264358.

[DLS01] Dennis Dams, Yassine Lakhnech, and Martin Steffen. “Iterating Transducers”. In: *International Conference on Computer Aided Verification, CAV*. Vol. 2102. Springer, 2001, pp. 286–297. DOI: 10.1007/3-540-44585-4_27.

[DLS02] Dennis Dams, Yassine Lakhnech, and Martin Steffen. “Iterating transducers”. In: *Journal of Logic and Algebraic Programming* 52-53 (2002), pp. 109–127. DOI: 10.1016/S1567-8326(02)00025-5.

[Doh+22] Taylor Dohmen, Noah Topper, George Atia, Andre Beckus, Ashutosh Trivedi, and Alvaro Velasquez. “Inferring Probabilistic Reward Machines from Non-Markovian Reward Signals for Reinforcement Learning”. In: *International Conference on Automated Planning and Scheduling, ICAPS*. Vol. 32. June 2022, pp. 574–582. DOI: 10.1609/icaps.v32i1.19844.

[Doh+24] Taylor Dohmen, Mateo Perez, Fabio Somenzi, and Ashutosh Trivedi. “Regular Reinforcement Learning”. In: *International Conference on Computer Aided Verification, CAV*. 2024.

[DT23] Taylor Dohmen and Ashutosh Trivedi. “Reinforcement Learning with Depreciating Assets”. In: *International Conference on Autonomous Agents and Multiagent Systems, AAMAS*. ACM, 2023, pp. 2628–2630. DOI: 10.5555/3545946.3599024.

[DV16] Parasara Sridhar Duggirala and Mahesh Viswanathan. “Parsimonious, Simulation Based Verification of Linear Systems”. In: *International Conference on Computer Aided Verification, CAV*. Vol. 9779. Springer, 2016, pp. 477–494. DOI: 10.1007/978-3-319-41528-4_26.

[EH01] Joost Engelfriet and Hendrik Jan Hoogeboom. “MSO definable string transductions and two-way finite-state transducers”. In: *ACM Transactions on Computational Logic* 2.2 (2001), pp. 216–254. DOI: 10.1145/371316.371512.

[EH07] Joost Engelfriet and Hendrik Jan Hoogeboom. “Finitary Compositions of Two-way Finite-State Transductions”. In: *Fundamenta Informaticae* 80.1-3 (2007), pp. 111–123. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi80-1-3-07.

[EH99] Joost Engelfriet and Hendrik Jan Hoogeboom. “Two-Way Finite State Transducers and Monadic Second-Order Logic”. In: *International Colloquium on Automata, Languages and Programming, ICALP*. Vol. 1644. Springer, 1999, pp. 311–320. DOI: 10.1007/3-540-48523-6_28.

[Eil74a] Samuel Eilenberg. *Automata, languages, and machines. A*. Pure and applied mathematics. Academic Press, 1974. ISBN: 0122340019.

[Eil74b] Samuel Eilenberg. “Sequential Machines”. In: *Automata, languages, and machines. A*. Academic Press, 1974. Chap. 11, pp. 296–329. ISBN: 0122340019.

[Elg61] Calvin C. Elgot. “Decision Problems of Finite Automata Design and Related Arithmetics”. In: *Transactions of the American Mathematical Society* (1961), pp. 21–51. DOI: 10.1090/S0002-9947-1961-0139530-9.

[EM65] Calvin C. Elgot and Jorge E. Mezei. “On Relations Defined by Generalized Finite Automata”. In: *IBM Journal of Research and Development* 9.1 (1965), pp. 47–68. DOI: 10.1147/rd.91.0047.

[EW00] Kousha Etessami and Thomas Wilke. “An Until Hierarchy and Other Applications of an Ehrenfeucht-Fraïssé Game for Temporal Logic”. In: *Information and Computation* 160.1-2 (2000), pp. 88–108. DOI: 10.1006/inco.1999.2846.

[EY10] Kousha Etessami and Mihalis Yannakakis. “On the Complexity of Nash Equilibria and Other Fixed Points”. In: *SIAM Journal on Computing* 39.6 (2010), pp. 2531–2597. DOI: 10.1137/080720826.

[Fei11] Eugene A. Feinberg. *Total Expected Discounted Reward MDPS: Existence of Optimal Policies*. 2011. DOI: 10.1002/9780470400531.eorms0906.

[FGL19] Emmanuel Filiot, Olivier Gauwin, and Nathan Lhote. “Logical and Algebraic Characterizations of Rational Transductions”. In: *Logical Methods in Computer Science* 15, Issue 4 (Dec. 2019). ISSN: 1860-5974. DOI: 10.23638/LMCS-15(4:16)2019. eprint: 1705.03726.

[FHZ11] Oliver Friedmann, Thomas Dueholm Hansen, and Uri Zwick. “Subexponential lower bounds for randomized pivoting rules for the simplex algorithm”. In: *Symposium on Theory of Computing, STOC*. ACM, 2011, pp. 283–292. DOI: 10.1145/1993636.1993675.

[Fil+18] Emmanuel Filiot, Olivier Gauwin, Nathan Lhote, and Anca Muscholl. “On Canonical Models for Rational Functions over Infinite Words”. In: *Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*. Vol. 122. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 30:1–30:17. DOI: 10.4230/LIPIcs.FSTTCS.2018.30.

[Fil15] Emmanuel Filiot. “Logic-Automata Connections for Transformations”. In: *Logic and Its Applications*. Springer, 2015, pp. 30–57. ISBN: 978-3-662-45824-2. DOI: 10.1007/978-3-662-45824-2_3.

[FLO02] Shane Frederick, George Loewenstein, and Ted O’Donoghue. “Time Discounting and Time Preference: A Critical Review”. In: *Journal of Economic Literature* 40.2 (June 2002), pp. 351–401. DOI: 10.1257/002205102320161311.

[FR16] Emmanuel Filiot and Pierre-Alain Reynier. “Transducers, logic and algebra for functions of finite words”. In: *ACM SIGLOG News* 3.3 (Aug. 2016), pp. 4–19. ISSN: 2372-3491. DOI: 10.1145/2984450.2984453.

[FR17] Emmanuel Filiot and Pierre-Alain Reynier. “Copyful Streaming String Transducers”. In: *Reachability Problems, RP*. Vol. 10506. Springer, 2017, pp. 75–86. DOI: 10.1007/978-3-319-67089-8_6.

[FR21] Emmanuel Filiot and Pierre-Alain Reynier. “Copyful Streaming String Transducers”. In: *Fundamenta Informaticae* 178.1-2 (2021), pp. 59–76. DOI: 10.3233/FI-2021-1998.

[FS01] Eugene A Feinberg and Adam Shwartz. *Handbook of Markov decision processes: methods and applications*. Springer, 2001. DOI: 10.1007/978-1-4615-0805-2.

[FTV02] János Flesch, Frank Thuijsman, and O. J. Vrieze. “Optimality in different strategy classes in zero-sum stochastic games”. In: *Mathematical Methods of Operations Research* 56.2 (2002), pp. 315–322. DOI: 10.1007/s001860200220.

[FV96] Jerzy Filar and Koos Vrieze. *Competitive Markov decision processes*. Springer-Verlag, 1996. DOI: 10.1007/978-1-4612-4054-9.

[Gab87] Dov M. Gabbay. “The Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems”. In: *Temporal Logic in Specification*. Vol. 398. Springer, 1987, pp. 409–448. DOI: 10.1007/3-540-51803-7_36.

[GB20] Maor Gaon and Ronen I. Brafman. “Reinforcement Learning with Non-Markovian Rewards”. In: *Conference on Artificial Intelligence, AAAI*. AAAI Press, 2020, pp. 3980–3987. DOI: 10.1609/aaai.v34i04.5814.

[Geh+18] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. “AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation”. In: *Symposium on Security and Privacy, SP*. IEEE, 2018, pp. 3–18. DOI: 10.1109/SP.2018.00058.

[GGP09] Khalil Ghorbal, Eric Goubault, and Sylvie Putot. “The Zonotope Abstract Domain Taylor1+”. In: *International Conference on Computer Aided Verification, CAV*. Vol. 5643. Springer, 2009, pp. 627–633. DOI: 10.1007/978-3-642-02658-4_47.

[Gia+19] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. “Foundations for Restraining Bolts: Reinforcement Learning with LTLf/LDLf Restraining Specifications”. In: *International Conference on Automated Planning and Scheduling, ICAPS*. AAAI Press, 2019, pp. 128–136. DOI: 10.1609/icaps.v29i1.3549.

[Gia+20] Giuseppe De Giacomo, Antonio Di Stasio, Francesco Fuggitti, and Sasha Rubin. “Pure-Past Linear Temporal and Dynamic Logic on Finite Traces”. In: *International Joint Conference on Artificial Intelligence, IJCAI*. ijcai.org, 2020, pp. 4959–4965. DOI: 10.24963/ijcai.2020/690.

[Gil58] Dean Gillette. “Stochastic Games With Zero Stop Probabilities”. In: *Contributions to the Theory of Games (AM-39), Volume III*. Princeton University Press, 1958, pp. 179–188. DOI: 10.1515/9781400882151-011.

[Gim07] Hugo Gimbert. “Pure Stationary Optimal Strategies in Markov Decision Processes”. In: *Symposium on Theoretical Aspects of Computer Science, STACS*. Vol. 4393. Springer, 2007, pp. 200–211. DOI: 10.1007/978-3-540-70918-3_18.

[GKR17] Valentin Goranko, Antti Kuusisto, and Raine Rönnholm. “Game-Theoretic Semantics for ATL+ with Applications to Model Checking”. In: *Conference on Autonomous Agents and MultiAgent Systems, AAMAS*. ACM, 2017, pp. 1277–1285. DOI: 10.5555/3091125.3091302.

[GKR21] Valentin Goranko, Antti Kuusisto, and Raine Rönnholm. “Game-theoretic semantics for ATL^+ with applications to model checking”. In: *Information and Computation* 276 (2021), p. 104554. DOI: 10.1016/j.ic.2020.104554.

[GMP16] Vincenzo Del Giudice, Benedetto Manganelli, and Pierfrancesco De Paola. “Depreciation Methods for Firm’s Assets”. In: *Computational Science and Its Applications - ICCSA*. Vol. 9788. Springer, 2016, pp. 214–227. DOI: 10.1007/978-3-319-42111-7_17.

[GR66] Seymour Ginsburg and Gene F. Rose. “A Characterization of Machine Mappings”. In: *Canadian Journal of Mathematics* 18 (Jan. 1966), pp. 381–388. ISSN: 0008-414X. DOI: 10.4153/CJM-1966-040-3.

[GRF00] Dov M Gabbay, Mark A Reynolds, and Marcelo Finger. *Temporal logic: mathematical foundations and computational aspects*. Oxford University Press, 2000. DOI: 10.1093/oso/9780198537687.001.0001.

[Gri68] Timothy V. Griffiths. “The Unsolvability of the Equivalence Problem for Lambda-Free Nondeterministic Generalized Machines”. In: *Journal of the ACM* 15.3 (1968), pp. 409–413. DOI: 10.1145/321466.321473.

[GT88] Andrew V. Goldberg and Robert Endre Tarjan. “Finding Minimum-Cost Circulations by Canceling Negative Cycles”. In: *Symposium on Theory of Computing, STOC*. ACM, 1988, pp. 388–397. DOI: 10.1145/62212.62250.

[GT89] Andrew V. Goldberg and Robert Endre Tarjan. “Finding minimum-cost circulations by canceling negative cycles”. In: *Journal of the ACM* 36.4 (1989), pp. 873–886. DOI: 10.1145/76359.76368.

[Gur80] Eitan M. Gurari. “The Equivalence Problem for Deterministic Two-Way Sequential Transducers Is Decidable”. In: *Symposium on Foundations of Computer Science, FOCS*. IEEE, 1980, pp. 83–85. DOI: 10.1109/SFCS.1980.46.

[Gur82] Eitan M. Gurari. “The Equivalence Problem for Deterministic Two-Way Sequential Transducers is Decidable”. In: *SIAM Journal on Computing* 11.3 (1982), pp. 448–452. DOI: 10.1137/0211035.

[GV13] Giuseppe De Giacomo and Moshe Y. Vardi. “Linear Temporal Logic and Linear Dynamic Logic on Finite Traces”. In: *International Joint Conference on Artificial Intelligence, IJCAI*. IJCAI/AAAI, 2013, pp. 854–860. DOI: 10.5555/2540128.2540252.

[GV15] Giuseppe De Giacomo and Moshe Y. Vardi. “Synthesis for LTL and LDL on Finite Traces”. In: *International Joint Conference on Artificial Intelligence, IJCAI*. AAAI Press, 2015, pp. 1558–1564. DOI: 10.5555/2832415.2832466.

[Hah+19a] Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. "Limit reachability for model-free reinforcement learning of ω-regular objectives". In: *International Workshop on Symbolic-Numeric methods for Reasoning about CPS and IoT, SNR*. ACM, 2019, pp. 16–18. DOI: 10.1145/3313149.3313369.

[Hah+19b] Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. "Omega-Regular Objectives in Model-Free Reinforcement Learning". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS*. Vol. 11427. Springer, 2019, pp. 395–412. DOI: 10.1007/978-3-030-17462-0_27.

[Hah+20] Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. "Faithful and Effective Reward Schemes for Model-Free Reinforcement Learning of Omega-Regular Objectives". In: *International Symposium on Automated Technology for Verification and Analysis, ATVA*. Vol. 12302. Springer, 2020, pp. 108–124. DOI: 10.1007/978-3-030-59152-6_6.

[Hal07] Nir Halman. "Simple Stochastic Games, Parity Games, Mean Payoff Games and Discounted Payoff Games Are All LP-Type Problems". In: *Algorithmica* 49.1 (2007), pp. 37–50. DOI: 10.1007/s00453-007-0175-3.

[Han+11] Kristoffer Arnsfelt Hansen, Michal Koucký, Niels Lauritzen, Peter Bro Miltersen, and Elias P. Tsigaridas. "Exact algorithms for solving stochastic games: extended abstract". In: *Symposium on Theory of Computing, STOC*. ACM, 2011, pp. 205–214. DOI: 10.1145/1993636.1993665.

[Hea07] Geoffrey Heal. "Discounting: A Review of the Basic Economics". In: *The University of Chicago Law Review* 74.1 (2007), pp. 59–77. URL: http://www.jstor.org/stable/4495597.

[Hen+23] Thomas A. Henzinger, Pavol Kebis, Nicolas Mazzocchi, and N. Ege Saraç. "Regular Methods for Operator Precedence Languages". In: *International Colloquium on Automata, Languages, and Programming, ICALP*. Vol. 261. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 129:1–129:20. DOI: 10.4230/LIPIcs.ICALP.2023.129.

[HI13] Thomas Dueholm Hansen and Rasmus Ibsen-Jensen. "The Complexity of Interior Point Methods for Solving Discounted Turn-Based Stochastic Games". In: *Conference on Computability in Europe, CiE*. Vol. 7921. Springer, 2013, pp. 252–262. DOI: 10.1007/978-3-642-39053-1_29.

[HK66] Alan J. Hoffman and Richard M. Karp. "On nonterminating stochastic games". In: *Management Science* 12.5 (1966), pp. 359–370. DOI: 10.1287/mnsc.12.5.359.

[HKR20] Lauri Hella, Antti Kuusisto, and Raine Rönnholm. "Bounded Game-Theoretic Semantics for Modal Mu-Calculus and Some Variants". In: *International Symposium on Games, Automata, Logics, and Formal Verification, GandALF*. Vol. 326. 2020, pp. 82–96. DOI: 10.4204/EPTCS.326.6.

[HKR22] Lauri Hella, Antti Kuusisto, and Raine Rönnholm. "Bounded game-theoretic semantics for modal mu-calculus". In: *Information and Computation* 289.Part (2022), p. 104882. DOI: 10.1016/j.ic.2022.104882.

[HL20] Patrick Henriksen and Alessio R. Lomuscio. "Efficient Neural Network Verification via Adaptive Refinement and Adversarial Search". In: *European Conference on Artificial Intelligence, ECAI*. Vol. 325. IOS Press, 2020, pp. 2513–2520. DOI: 10.3233/FAIA200385.

[HL21] Patrick Henriksen and Alessio Lomuscio. "DEEPSPLIT: An Efficient Splitting Method for Neural Network Verification via Indirect Effect Analysis". In: *International Joint Conference on Artificial Intelligence, IJCAI*. ijcai.org, 2021, pp. 2549–2555. DOI: 10.24963/ijcai.2021/351.

[HMU07] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley, 2007. ISBN: 978-0-321-47617-3.

[HMZ13] Thomas Dueholm Hansen, Peter Bro Miltersen, and Uri Zwick. "Strategy Iteration Is Strongly Polynomial for 2-Player Turn-Based Stochastic Games with a Constant Discount Factor". In: *Journal of the ACM* 60.1 (2013), 1:1–1:16. DOI: 10.1145/2432622.2432623.

[Hop71] John Hopcroft. "AN n log n ALGORITHM FOR MINIMIZING STATES IN A FINITE AUTOMATON". In: *Theory of Machines and Computations*. Academic Press, 1971, pp. 189–196. ISBN: 978-0-12-417750-5. DOI: 10.1016/B978-0-12-417750-5.50022-1.

[Hot25] Harold Hotelling. "A General Mathematical Theory of Depreciation". In: *Journal of the American Statistical Association* 20.151 (1925), pp. 340–353. DOI: 10.2307/2965518.

[HRS05] Aidan Harding, Mark Ryan, and Pierre-Yves Schobbens. "A New Algorithm for Strategy Synthesis in LTL Games". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS*. Vol. 3440. Springer, 2005, pp. 477–492. DOI: 10.1007/978-3-540-31980-1_31.

[Huf54] D.A. Huffman. "The synthesis of sequential switching circuits". In: *Journal of the Franklin Institute* 257.3 (1954), pp. 161–190. ISSN: 0016-0032. DOI: 10.1016/0016-0032(54)90574-8.

[HV04] Peter Habermehl and Tomás Vojnar. "Regular Model Checking Using Inference of Regular Languages". In: *International Workshop on Verification of Infinite-State Systems, INFINITY*. Elsevier, 2004, pp. 21–36. DOI: 10.1016/j.entcs.2005.01.044.

[HW80] Charles R Hulten and Frank C Wykoff. *The measurement of economic depreciation*. Urban Institute Washington, 1980.

[HW96] Charles R. Hulten and Frank C. Wykoff. "Issues in the measurement of economic depreciation: Introductory remarks". In: *Economic Inquiry* 34.1 (Jan. 1996), p. 10. DOI: 10.1111/j.1465-7295.1996.tb01361.x.

[Ica+18] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. "Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning". In: *International Conference on Machine Learning, ICML*. Vol. 80. PMLR, 2018, pp. 2112–2121. URL: http://proceedings.mlr.press/v80/icarte18a.html.

[Ica+19] Rodrigo Toro Icarte, Ethan Waldie, Toryn Q. Klassen, Richard Anthony Valenzano, Margarita P. Castro, and Sheila A. McIlraith. "Learning Reward Machines for Partially Observable Reinforcement Learning". In: *Conference on Neural Information Processing Systems, NeurIPS*. 2019, pp. 15497–15508. URL: https://proceedings.neurips.cc/paper/2019/hash/532435c44bec236b471a47a88d63513d-Abstract.html.

[Ica+22] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. "Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning". In: *Journal of Artificial Intelligence Research* 73 (2022), pp. 173–208. DOI: 10.1613/jair.1.12440.

[Ica22] Rodrigo Toro Icarte. “Reward Machines”. PhD thesis. University of Toronto, Canada, 2022. URL: http://hdl.handle.net/1807/110754.

[JN00] Bengt Jonsson and Marcus Nilsson. “Transitive Closures of Regular Relations for Verifying Infinite-State Systems”. In: *International Conference on Tools and Algorithms for Construction and Analysis of Systems, TACAS*. Vol. 1785. Springer, 2000, pp. 220–234. DOI: 10.1007/3-540-46419-0_16.

[Kat+17] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. “Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks”. In: *International Conference on Computer Aided Verification, CAV*. Vol. 10426. Springer, 2017, pp. 97–117. DOI: 10.1007/978-3-319-63387-9_5.

[Kat+19] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. “The Marabou Framework for Verification and Analysis of Deep Neural Networks”. In: *International Conference on Computer Aided Verification, CAV*. Vol. 11561. Springer, 2019, pp. 443–452. DOI: 10.1007/978-3-030-25540-4_26.

[Kaz+22] Milad Kazemi, Mateo Perez, Fabio Somenzi, Sadegh Soudjani, Ashutosh Trivedi, and Alvaro Velasquez. “Translating Omega-Regular Specifications to Average Objectives for Model-Free Reinforcement Learning”. In: *International Conference on Autonomous Agents and Multiagent Systems, AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2022, pp. 732–741. DOI: 10.5555/3535850.3535933.

[Kes+01] Yonit Kesten, Oded Maler, Monica Marcus, Amir Pnueli, and Elad Shahar. “Symbolic model checking with rich assertional languages”. In: *Theoretical Computer Science* 256.1-2 (2001), pp. 93–112. DOI: 10.1016/S0304-3975(00)00103-1.

[Kha79] Leonid Genrikhovich Khachiyan. “A polynomial algorithm in linear programming”. In: *Doklady Akademii Nauk*. Vol. 244. Russian Academy of Sciences. 1979, pp. 1093–1096.

[Kle56] Stephen Cole Kleene. “Representation of Events in Nerve Nets and Finite Automata”. In: *Automata Studies. (AM-34), Volume 34*. Princeton University Press, 1956, pp. 3–42. ISBN: 9781400882618. DOI: 10.1515/9781400882618-002.

[KM72] Victor Klee and George J Minty. “How good is the simplex algorithm”. In: *Inequalities* 3.3 (1972), pp. 159–175.

[KW17] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *International Conference on Learning Representations, ICLR*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=SJU4ayYgl.

[Lam83] Leslie Lamport. “What Good is Temporal Logic?” In: *IFIP World Computer Congress*. North-Holland/IFIP, 1983, pp. 657–668.

[Lan05] Martin Lange. “A quick axiomatisation of LTL with past”. In: *Mathematical Logic Quarterly* 51.1 (2005), pp. 83–88. DOI: 10.1002/malq.200410009.

[Law04] Mark V. Lawson. *Finite Automata*. Chapman and Hall/CRC, Apr. 2004. ISBN: 978-0-42917574-9. DOI: 10.1201/9781482285840.

[Leg12] Axel Legay. “Extrapolating (omega-)regular model checking”. In: *International Journal on Software Tools for Technology Transfer* 14.2 (2012), pp. 119–143. DOI: 10.1007/s10009-011-0209-7.

[Liu+21] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher A. Strong, Clark W. Barrett, and Mykel J. Kochenderfer. "Algorithms for Verifying Deep Neural Networks". In: *Foundational Trends in Optimization* 4.3-4 (2021), pp. 244–404. DOI: 10.1561/2400000035.

[LJ92] George Loewenstein and Elster Jon, eds. *Choice Over Time*. Russell Sage Foundation, 1992, p. 424.

[LL69] Thomas M. Liggett and Steven A. Lippman. "Stochastic Games with Perfect Information and Time Average Payoff". In: *SIAM Review* 11.4 (1969), pp. 604–607. DOI: 10.1137/1011093.

[LM05] Shuvendu K. Lahiri and Madanlal Musuvathi. "An Efficient Decision Procedure for UTVPI Constraints". In: *International Workshop on Frontiers of Combining Systems, FroCoS*. Vol. 3717. Springer, 2005, pp. 168–183. DOI: 10.1007/11559306_9.

[LMT19] Théodore Lopez, Benjamin Monmege, and Jean-Marc Talbot. "Determinisation of Finitely-Ambiguous Copyless Cost Register Automata". In: *International Symposium on Mathematical Foundations of Computer Science, MFCS*. Vol. 138. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 75:1–75:15. DOI: 10.4230/LIPIcs.MFCS.2019.75.

[LN73] Hans Lausch and Wilfried Nöbauer. "Composition of Polynomials and Polynomial Functions Over Rings and Fields". In: *Algebra of Polynomials*. Elsevier, 1973. Chap. 4, pp. 134–222. DOI: 10.1016/S0924-6509(08)70606-8.

[LPZ85] Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. "The Glory of the Past". In: *Logics of Programs, Conference*. Vol. 193. Springer, 1985, pp. 196–218. DOI: 10.1007/3-540-15648-8_16.

[LS94] François Laroussinie and Philippe Schnoebelen. "A Hierarchy of Temporal Logics with Past (Extended Abstract)". In: *Symposium on Theoretical Aspects of Computer Science, STACS*. Vol. 775. Springer, 1994, pp. 47–58. DOI: 10.1007/3-540-57785-8_130.

[LS95] François Laroussinie and Philippe Schnoebelen. "A Hierarchy of Temporal Logics with Past". In: *Theoretical Computer Science* 148.2 (1995), pp. 303–324. DOI: 10.1016/0304-3975(95)00035-U.

[LW10] Axel Legay and Pierre Wolper. "On (Omega-)regular model checking". In: *ACM Transactions on Computational Logic* 12.1 (2010), 2:1–2:46. DOI: 10.1145/1838552.1838554.

[Man12] Eleni Mandrali. "Weighted LTL with Discounting". In: *International Conference on Implementation and Application of Automata, CIAA*. Vol. 7381. Springer, 2012, pp. 353–360. DOI: 10.1007/978-3-642-31606-7_32.

[Mar02] Nicolas Markey. "Past is for Free: on the Complexity of Verifying Linear Temporal Properties with Past". In: *International Workshop on Expressiveness in Concurrency, EXPRESS*. Vol. 68. Elsevier, 2002, pp. 87–104. DOI: 10.1016/S1571-0661(05)80366-4.

[Mar03] Nicolas Markey. "Temporal Logic with Past is Exponentially More Succinct". In: *Bulletin - European Association for Theoretical Computer Science* 79 (2003), pp. 122–128. URL: https://hal.science/hal-01194627.

[Mar04] Nicolas Markey. "Past is for free: on the complexity of verifying linear temporal properties with past". In: *Acta Informatica* 40.6-7 (2004), pp. 431–458. DOI: 10.1007/s00236-003-0136-5.

[MB08] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. "Z3: An Efficient SMT Solver". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS*. Vol. 4963. Springer, 2008, pp. 337–340. DOI: 10.1007/978-3-540-78800-3_24.

[McN66] Robert McNaughton. "Testing and Generating Infinite Sequences by a Finite Automaton". In: *Information and Control* 9.5 (1966), pp. 521–530. DOI: 10.1016/S0019-9958(66)80013-X.

[Mea55] George H. Mealy. "A method for synthesizing sequential circuits". In: *Bell System Technical Journal* 34.5 (Sept. 1955), pp. 1045–1079. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1955.tb03788.x.

[Mey06] Albert R. Meyer. "Weak monadic second order theory of succesor is not elementary-recursive". In: *Logic Colloquium*. Berlin, Germany: Springer, Aug. 2006, pp. 132–154. ISBN: 978-3-540-37483-1. DOI: 10.1007/BFb0064872.

[Mil02] Emanuel Milman. "The Semi-Algebraic Theory of Stochastic Games". In: *Mathematics of Operations Research* 27.2 (2002), pp. 401–418. DOI: 10.1287/moor.27.2.401.320.

[Min01] Antoine Miné. "The Octagon Abstract Domain". In: *Working Conference on Reverse Engineering, WCRE*. IEEE, 2001, p. 310. DOI: 10.1109/WCRE.2001.957836.

[Min06] Antoine Miné. "The octagon abstract domain". In: *Higher Order Symbolic Computation* 19.1 (2006), pp. 31–100. DOI: 10.1007/s10990-006-8609-1.

[MN81] J.F. Mertens and A. Neyman. "Stochastic Games". In: *International Journal of Game Theory* 10.2 (June 1981), pp. 53–66. ISSN: 0020-7276. DOI: 10.1007/BF01769259.

[Moo56] Edward F Moore. "Gedanken-Experiments on Sequential Machines". In: *Automata Studies*. Annals of Mathematics Studies (1956), pp. 129–153.

[MP19] Anca Muscholl and Gabriele Puppis. "Equivalence of Finite-Valued Streaming String Transducers Is Decidable". In: *International Colloquium on Automata, Languages, and Programming, ICALP*. Vol. 132. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 122:1–122:15. DOI: 10.4230/LIPIcs.ICALP.2019.122.

[MP70] A Maitra and T Parthasarathy. "On stochastic games". In: *Journal of Optimization Theory and Applications* 5 (1970), pp. 289–300. DOI: 10.1007/BF00927915.

[MS73] AR Meyer and L Stockmeyer. "Nonelementary word problems in automata and logic". In: *AMS symposium on complexity of computation*. 1973, pp. 38–49.

[MT02] Jerzy Marcinkowski and Tomasz Truderung. "Optimal Complexity Bounds for Positive LTL Games". In: *International Workshop on Computer Science Logic, CSL*. Vol. 2471. Springer, 2002, pp. 262–275. DOI: 10.1007/3-540-45793-3_18.

[Mul63] David E. Muller. "Infinite sequences and finite machines". In: *Symposium on Switching Circuit Theory and Logical Design*. IEEE, 1963, pp. 3–16. DOI: 10.1109/SWCT.1963.8.

[MY60] Robert McNaughton and Hisao Yamada. "Regular Expressions and State Graphs for Automata". In: *IRE Transactions on Electronic Computers* 9.1 (1960), pp. 39–47. DOI: 10.1109/TEC.1960.5221603.

[Myh57] John Myhill. "Finite automata and the representation of events". In: *WADD Technical Report* 57 (1957), pp. 112–137.

[Nei+21] Daniel Neider, Jean-Raphaël Gaglione, Ivan Gavran, Ufuk Topcu, Bo Wu, and Zhe Xu. "Advice-Guided Reinforcement Learning in a non-Markovian Environment". In: *Conference on Artificial Intelligence, AAAI*. AAAI Press, 2021, pp. 9073–9080. DOI: 10.1609/aaai.v35i10.17096.

[Ner58] Anil Nerode. "Linear automaton transformations". In: *Proceedings of the American Mathematical Society* (1958), pp. 541–544. DOI: 10.2307/2033204.

[Nor98] James R. Norris. *Markov chains*. Cambridge series in statistical and probabilistic mathematics. Cambridge University Press, 1998. ISBN: 978-0-521-48181-6.

[NS03] Abraham Neyman and Sylvain Sorin. *Stochastic games and applications*. Vol. 570. Springer, 2003. DOI: 10.1007/978-94-010-0189-2.

[Oli14] Miquel Oliu-Barton. "The Asymptotic Value in Finite Stochastic Games". In: *Mathematics of Operations Research* 39.3 (2014), pp. 712–721. DOI: 10.1287/moor.2013.0642.

[Oli21] Miquel Oliu-Barton. "New Algorithms for Solving Zero-Sum Stochastic Games". In: *Mathematics of Operations Research* 46.1 (2021), pp. 255–267. DOI: 10.1287/moor.2020.1055.

[Orl96] James B. Orlin. "A Polynomial Time Primal Network Simplex Algorithm for Minimum Cost Flows (An Extended Abstract)". In: *Symposium on Discrete Algorithms, SODA*. ACM/SIAM, 1996, pp. 474–481. DOI: 10.5555/313852.314105.

[Orl97] James B. Orlin. "A polynomial time primal network simplex algorithm for minimum cost flows". In: *Mathematical Programming* 77 (1997), pp. 109–129. DOI: 10.1007/BF02614365.

[Pan+13] Federica Panella, Matteo Pradella, Violetta Lonati, and Dino Mandrioli. "Operator Precedence ω-Languages". In: *International Conference on Developments in Language Theory, DLT*. Vol. 7907. Springer, 2013, pp. 396–408. DOI: 10.1007/978-3-642-38771-5_35.

[Pan16] Federica Panella. "Operator precedence languages: theory and applications". PhD thesis. Polytechnic University of Milan, Italy, 2016. URL: https://hdl.handle.net/10589/114646.

[Phi99] Cédric Philibert. "The economics of climate change and the theory of discounting". In: *Energy Policy* 27.15 (1999), pp. 913–927. DOI: 10.1016/S0301-4215(99)00081-6.

[Pla22] Aske Plaat. *Deep Reinforcement Learning*. Springer, 2022. ISBN: 978-981-19-0638-1. DOI: 10.1007/978-981-19-0638-1.

[Pnu77] Amir Pnueli. "The Temporal Logic of Programs". In: *Symposium on Foundations of Computer Science, FOCS*. IEEE, 1977, pp. 46–57. DOI: 10.1109/SFCS.1977.32.

[PP04] Dominique Perrin and Jean-Eric Pin. *Infinite words - automata, semigroups, logic and games*. Vol. 141. Pure and applied mathematics series. Elsevier Morgan Kaufmann, 2004. ISBN: 978-0-12-532111-2. URL: https://www.sciencedirect.com/bookseries/pure-and-applied-mathematics/vol/141.

[Pre38] Gabriel A. D. Preinreich. "Annual Survey of Economic Theory: The Theory of Depreciation". In: *Econometrica* 6.3 (1938), pp. 219–241. DOI: 10.2307/1907053.

[Put94] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994. ISBN: 978-0-47161977-2. DOI: 10.1002/9780470316887.

[RCN73] Singiresu S Rao, R Chandrasekaran, and KPK Nair. "Algorithms for discounted stochastic games". In: *Journal of Optimization Theory and Applications* 11.6 (1973), pp. 627–637. DOI: 10.1007/BF00935562.

[Ren+21] Gavin Rens, Jean-François Raskin, Raphaël Reynouard, and Giuseppe Marra. "Online Learning of non-Markovian Reward Models". In: *International Conference on Agents and Artificial Intelligence, ICAART*. SCITEPRESS, 2021, pp. 74–86. DOI: 10.5220/0010212000740086.

[RF91] T. E. S. Raghavan and Jerzy A. Filar. "Algorithms for stochastic games - A survey". In: *Zeitschrift für Operations Research - Methods and Models of Operations Research* 35.6 (1991), pp. 437–472. DOI: 10.1007/BF01415989.

[Ric53] Henry Gordon Rice. "Classes of recursively enumerable sets and their decision problems". In: *Transactions of the American Mathematical society* 74.2 (1953), pp. 358–366.

[RM51] Herbert Robbins and Sutton Monro. "A Stochastic Approximation Method". In: *Annals of Mathematical Statistics* 22.3 (Sept. 1951), pp. 400–407. ISSN: 0003-4851. DOI: 10.1214/aoms/1177729586.

[RS03] T. E. S. Raghavan and Zamir Syed. "A policy-improvement type algorithm for solving zero-sum two-person stochastic games of perfect information". In: *Mathematical Programming* 95.3 (2003), pp. 513–532. DOI: 10.1007/s10107-002-0312-3.

[RS59] Michael O. Rabin and Dana S. Scott. "Finite Automata and Their Decision Problems". In: *IBM Journal of Research and Development* 3.2 (1959), pp. 114–125. DOI: 10.1147/rd.32.0114.

[RS91] Christophe Reutenauer and Marcel-Paul Schutzenberger. "Minimization of Rational Word Functions". In: *SIAM Journal on Computing* 20.4 (1991), pp. 669–685. DOI: 10.1137/0220042.

[RU71] Nicholas Rescher and Alasdair Urquhart. *Temporal logic*. Vol. 3. Springer, 1971. DOI: 10.1007/978-3-7091-7664-1.

[Rud76] W. Rudin. *Principles of Mathematical Analysis*. International series in pure and applied mathematics. McGraw-Hill, 1976. ISBN: 9780070856134.

[Sak09] Jacques Sakarovitch. *Elements of automata theory*. Cambridge University Press, 2009.

[Sav70] Walter J. Savitch. "Relationships Between Nondeterministic and Deterministic Tape Complexities". In: *Journal of Computer and System Sciences* 4.2 (1970), pp. 177–192. DOI: 10.1016/S0022-0000(70)80006-X.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998. ISBN: 978-0-262-19398-6. URL: https://www.worldcat.org/oclc/37293240.

[Sch+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). DOI: 10.48550/arXiv.1707.06347. arXiv: 1707.06347.

[Sch61] M. P. Schtzenberger. "A remark on finite transducers". In: *Information and Control* 4.2 (Sept. 1961), pp. 185–196. ISSN: 0019-9958. DOI: 10.1016/S0019-9958(61)80006-5.

[Sch76] Marcel Paul Schützenberger. "Sur les Relations Rationnelles Entre Monoides Libres". In: *Theoretical Computer Science* 3.2 (1976), pp. 243–259. DOI: 10.1016/0304-3975(76)90026-8.

[Sha53] Lloyd S Shapley. "Stochastic games". In: *Proceedings of the national academy of sciences* 39.10 (1953), pp. 1095–1100. DOI: 10.1073/pnas.39.10.1095.

[Sha81] M. Sharir. "A strong-connectivity algorithm and its applications in data flow analysis". In: *Computers and Mathematics with Applications* 7.1 (1981), pp. 67–72. ISSN: 0898-1221. DOI: 10.1016/0898-1221(81)90008-0.

[She59] John C. Shepherdson. "The Reduction of Two-Way Automata to One-Way Automata". In: *IBM Journal of Research and Development* 3.2 (1959), pp. 198–200. DOI: 10.1147/rd.32.0198.

[SHL65] Richard Edwin Stearns, Juris Hartmanis, and Philip M. Lewis. "Hierarchies of memory limited computations". In: *Symposium on Switching Circuit Theory and Logical Design*. IEEE, 1965, pp. 179–190. DOI: 10.1109/FOCS.1965.11.

[SHR13] Vikas Vikram Singh, N. Hemachandra, and K. S. Mallikarjuna Rao. "Blackwell Optimality in stochastic Games". In: *International Game Theory Review, IGTR* 15.4 (2013). DOI: 10.1142/S0219198913400252.

[Sid+20] Aaron Sidford, Mengdi Wang, Lin Yang, and Yinyu Ye. "Solving Discounted Stochastic Two-Player Games with Near-Optimal Time and Sample Complexity". In: *International Conference on Artificial Intelligence and Statistics, AISTATS*. Vol. 108. PMLR, 2020, pp. 2992–3002. URL: http://proceedings.mlr.press/v108/sidford20a.html.

[Sin+18] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. "Fast and Effective Robustness Certification". In: *Conference on Neural Information Processing Systems, NeurIPS*. 2018, pp. 10825–10836. URL: https://proceedings.neurips.cc/paper/2018/hash/f2f446980d8e971ef3da97af089481c3-Abstract.html.

[Sin+19] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. "An abstract domain for certifying neural networks". In: *ACM Proceedings on Programming Languages* 3.POPL (2019), 41:1–41:30. DOI: 10.1145/3290354.

[Sip97] Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997. ISBN: 978-0-534-94728-6.

[SM02] Larry Stockmeyer and Albert R. Meyer. "Cosmological lower bound on the circuit complexity of a small problem in logic". In: *Journal of the ACM* 49.6 (Nov. 2002), pp. 753–784. ISSN: 0004-5411. DOI: 10.1145/602220.602223.

[ST19] Fabio Somenzi and Ashutosh Trivedi. "Reinforcement Learning and Formal Requirements". In: *International Workshop on Numerical Software Verification, NSV@CAV*. Vol. 11652. Springer, 2019, pp. 26–41. DOI: 10.1007/978-3-030-28423-7_2.

[Sto74] Larry J. Stockmeyer. "The complexity of decision problems in automata theory and logic". PhD thesis. Massachusetts Institute of Technology, USA, 1974. URL: http://hdl.handle.net/1721.1/15540.

[Tap+19] Martin Tappler, Bernhard K. Aichernig, Giovanni Bacci, Maria Eichlseder, and Kim G. Larsen. "L^*-Based Learning of Markov Decision Processes". In: *World Congress on Formal Methods, FM*. Vol. 11800. Springer, 2019, pp. 651–669. DOI: 10.1007/978-3-030-30942-8_38.

[Tap+21] Martin Tappler, Bernhard K. Aichernig, Giovanni Bacci, Maria Eichlseder, and Kim G. Larsen. "L^*-based learning of Markov decision processes (extended version)". In: *Formal Aspects of Computing* 33.4-5 (2021), pp. 575–615. DOI: 10.1007/s00165-021-00536-5.

[Tar72] Robert Endre Tarjan. "Depth-First Search and Linear Graph Algorithms". In: *SIAM Journal on Computing* 1.2 (1972), pp. 146–160. DOI: 10.1137/0201010.

[Tay23] J. S. Taylor. "A Statistical Theory of Depreciation". In: *Journal of the American Statistical Association* 18.144 (1923), pp. 1010–1023. DOI: 10.2307/2965662.

[Tha67] J. W. Thatcher. "Characterizing derivation trees of context-free grammars through a generalization of finite automata theory". In: *Journal of Computer and System Sciences* 1.4 (Dec. 1967), pp. 317–322. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(67)80022-9.

[Tho90] Wolfgang Thomas. "Automata on Infinite Objects". In: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Elsevier and MIT Press, 1990, pp. 133–191. DOI: 10.1016/b978-0-444-88074-1.50009-3.

[Tho97] Wolfgang Thomas. "Languages, Automata, and Logic". In: *Handbook of Formal Languages, Volume 3: Beyond Words*. Springer, 1997, pp. 389–455. DOI: 10.1007/978-3-642-59126-6_7.

[Tou01] Tayssir Touili. "Regular Model Checking using Widening Techniques". In: *Workshop on Verification of Parameterized Systems, VEPAS*. Elsevier, 2001, pp. 342–356. DOI: 10.1016/S1571-0661(04)00187-2.

[Tra+19] Hoang-Dung Tran, Diego Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. "Star-Based Reachability Analysis of Deep Neural Networks". In: *World Congress on Formal Methods, FM*. Vol. 11800. Springer, 2019, pp. 670–686. DOI: 10.1007/978-3-030-30942-8_39.

[Tra+20a] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. "Verification of Deep Convolutional Neural Networks Using ImageStars". In: *International Conference on Computer Aided Verification, CAV*. Vol. 12224. Springer, 2020, pp. 18–42. DOI: 10.1007/978-3-030-53288-8_2.

[Tra+20b] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. "NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems". In: *International Conference on Computer Aided Verification, CAV*. Vol. 12224. Springer, 2020, pp. 3–17. DOI: 10.1007/978-3-030-53288-8_1.

[Tra+21] Hoang-Dung Tran, Neelanjana Pal, Patrick Musau, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Stanley Bak, and Taylor T. Johnson. "Robustness Verification of Semantic Segmentation Neural Networks Using Relaxed Reachability". In: *International Conference on Computer Aided Verification, CAV*. Vol. 12759. Springer, 2021, pp. 263–286. DOI: 10.1007/978-3-030-81685-8_12.

[Tra62a] Boris A. Trakhtenbrot. "Finite automata and monadic second order logic". In: *Siberian Mathematical Journal* (1962), pp. 103–131.

[Tra62b] Boris A. Trakhtenbrot. "Finite automata and the logic of one-place predicates". In: *American Mathematical Society Translations*. Vol. 59. American Mathematical Society, 1962, pp. 23–55.

[TXT19] Vincent Tjeng, Kai Yuanqing Xiao, and Russ Tedrake. "Evaluating Robustness of Neural Networks with Mixed Integer Programming". In: *International Conference on Learning Representations, ICLR*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=HyGIdiRqtm.

[Val84] Leslie G. Valiant. "A Theory of the Learnable". In: *Symposium on Theory of Computing, STOC*. ACM, 1984, pp. 436–445. DOI: 10.1145/800057.808710.

[Vel+21] Alvaro Velasquez, Brett Bissey, Lior Barak, Andre Beckus, Ismail Alkhouri, Daniel Melcer, and George K. Atia. "Dynamic Automaton-Guided Reward Shaping for Monte Carlo Tree Search". In: *Conference on Artificial Intelligence, AAAI*. AAAI Press, 2021, pp. 12015–12023. DOI: 10.1609/aaai.v35i13.17427.

[Ven04] G. Venkatesh. “Reasoning About Game Equilibria Using Temporal Logic”. In: *Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*. Vol. 3328. Springer, 2004, pp. 506–517. DOI: 10.1007/978-3-540-30538-5_42.

[Ven11] G. Venkatesh. “Temporal Logic with Preferences and Reasoning About Games”. In: *Proof, Computation and Agency - Logic at the Crossroads*. Vol. 352. Synthese library. Springer, 2011, pp. 241–258. DOI: 10.1007/978-94-007-0080-2_14.

[VM20] Alvaro Velasquez and Daniel Melcer. “Verification-Guided Tree Search”. In: *International Conference on Autonomous Agents and Multiagent Systems, AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 2026–2028. DOI: 10.5555/3398761.3399063.

[Wan+18] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. “Efficient Formal Safety Analysis of Neural Networks”. In: *Conference on Neural Information Processing Systems, NeurIPS*. 2018, pp. 6369–6379. URL: https://proceedings.neurips.cc/paper/2018/hash/2ecd2bd94734e5dd392d8678bc64cdab-Abstract.html.

[Wan+21] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. “Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Neural Network Robustness Verification”. In: *Conference on Neural Information Processing Systems, NeurIPS*. 2021, pp. 29909–29921. URL: https://proceedings.neurips.cc/paper/2021/hash/fac7fead96dafceaf80c1daffeae82a4-Abstract.html.

[WB98] Pierre Wolper and Bernard Boigelot. “Verifying Systems with Infinite but Regular State Spaces”. In: *International Conference on Computer Aided Verification, CAV*. Vol. 1427. Springer, 1998, pp. 88–97. DOI: 10.1007/BFb0028736.

[WD92] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8 (1992), pp. 279–292. DOI: 10.1007/BF00992698.

[Wil90] Herbert S Wilf. *Generating functionology*. Academic Press Professional, Inc., 1990.

[Wri64] F Kenneth Wright. “Towards a general theory of depreciation”. In: *Journal of accounting research* (1964), pp. 80–90. DOI: 10.2307/2490157.

[WT16] Min Wen and Ufuk Topcu. “Probably Approximately Correct Learning in Stochastic Games with Temporal Logic Specifications”. In: *International Joint Conference on Artificial Intelligence, IJCAI*. IJCAI/AAAI Press, 2016, pp. 3630–3636. URL: http://www.ijcai.org/Abstract/16/511.

[Wu+21] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. “A comprehensive survey on graph neural networks”. In: *Transactions on Neural Networks and Learning Systems* 32.1 (2021), pp. 4–24. DOI: 10.1109/TNNLS.2020.2978386.

[Wyk70] Frank C. Wykoff. “Capital Depreciation in the Postwar Period: Automobiles”. In: *The Review of Economics and Statistics* 52.2 (1970), pp. 168–172. DOI: 10.2307/1926117.

[Xu+20] Zhe Xu, Ivan Gavran, Yousef Ahmad, Rupak Majumdar, Daniel Neider, Ufuk Topcu, and Bo Wu. “Joint Inference of Reward Machines and Policies for Reinforcement Learning”. In: *International Conference on Automated Planning and Scheduling, ICAPS*. AAAI Press, 2020, pp. 590–598. DOI: 10.1609/icaps.v30i1.6756.

[Xu+21] Zhe Xu, Bo Wu, Aditya Ojha, Daniel Neider, and Ufuk Topcu. "Active Finite Reward Automaton Inference and Reinforcement Learning Using Queries and Counterexamples". In: *International Cross-Domain Conference on Machine Learning and Knowledge Extraction, CD-MAKE*. Vol. 12844. Springer, 2021, pp. 115–135. DOI: 10.1007/978-3-030-84060-0_8.

[Zil16a] Bruno Ziliotto. "A Tauberian Theorem for Nonexpansive Operators and Applications to Zero-Sum Stochastic Games". In: *Mathematics of Operations Research* 41.4 (2016), pp. 1522–1534. DOI: 10.1287/moor.2016.0788.

[Zil16b] Bruno Ziliotto. "General limit value in zero-sum stochastic games". In: *International Journal of Game Theory* 45.1-2 (2016), pp. 353–374. DOI: 10.1007/s00182-015-0509-3.

[Zil18] Bruno Ziliotto. "Tauberian theorems for general iterations of operators: Applications to zero-sum stochastic games". In: *Games and Economic Behavior* 108 (2018), pp. 486–503. DOI: 10.1016/j.geb.2018.01.009.

Appendices

Appendix A

Verification Of Neural Networks

Remark A.1

The material presented in this appendix is based on an FM 2023 paper *The Octatope Abstract Domain for Verification of Neural Networks* [Bak+23].

Abstract interpretation [CC77; Alb21] is a well-established framework for program verification that formalizes the exploration of the program semantics at the granularity provided by the underlying domain. For example, intervals [CC77] form an abstract domain facilitating analysis in which sets of states are represented as hyper-rectangles. Other abstract domains such as difference constraints, octagons (unit-two-variables-per-inequality or UTVPI), and polyhedral (linear constraints) have been successfully deployed for the verification of neural networks. However, the multi-layer architecture of neural networks, when combined with linear function composition followed by a non-linear activation function at each layer, results in the repeated intersection of abstract spaces with linear inequalities. For this reason, abstract domains that do not permit an efficient affine mapping suffer in exploring the layered state space of the neural networks.

Zonotopes [Sin+18] solve this problem by representing an abstract set as an affine mapping of an interval generator set. For zonotopes, the key operations for neural network verification, such as nonemptiness, optimization, and over-approximation, can be performed via efficient, enumerative procedures. Linear star sets [DV16; Tra+19] generalize zonotopes by representing the generator set using the polyhedral domains. This generalization, while improving the expressiveness, leads to the decision procedures depend upon solving linear programs, which tends to be the performance bottleneck in the

overall algorithm. While linear programming is known to be solvable in polynomial time, via a number of celebrated interior-point algorithms [Kha79], there is no known strongly polynomial algorithm. Dantzig's simplex algorithm is a popular algorithm to solve LP and works well in practice, but for general LPs, the time complexity of the simplex algorithm is not polynomial [KM72], and subexponential lower bounds hold even for randomized pivoting rules [FHZ11].

For some subclasses of linear programming problems, more efficient solutions exist. In particular, when the constraints are restricted to difference constraints, which are of the form $x_i - x_j \leq c$, or UTVPI constraints, which are of the form $\pm x_i \pm x_j \leq c$, then the duals of the corresponding LPs can be reduced to *minimum cost flow* (MCF) problems [AMO93], for which there exist strongly polynomial time algorithms [GT88; GT89]. The Out-of-Kilter algorithm is one popular algorithm for solving minimum cost flow that also produces a solution to the dual [AMO93]. It runs in time $O((m^2 + m \cdot n \cdot \log n) \cdot U)$ on a network with m arcs and n nodes and maximum supply/demand U. Alternatively, the network simplex algorithm is a specialized version of the simplex algorithm to solve minimum cost flow problems. Unlike standard simplex, network simplex runs in polynomial time [Orl96; Orl97]. Given its relative efficiency, it is natural to ask: *in neural network verification, is it possible to replace expensive linear programming with min-cost flow calls?*

This question motivates the investigation of subclasses of star sets that are more general than zonotopes, but enable efficient decision procedures based on MCF problems. For this purpose, we introduce octatopes: images of affine maps of UTVPI constrained sets (octagons [Min01; Min06]). Since octatopes are a special class of star sets, the affine transformation remains efficient. We also study hexatopes: affine images of difference constrained sets (hexagons [Min01; Min06] or zones [Beh+06]). A key contribution of this work is that the operations required for verification using octatopes and hexatopes can be performed efficiently using algorithms for MCF problems.

Given that the MCF problem can be solved efficiently via Out-of-Kilter algorithm and network simplex (touted [BJS09] to be 200-300 times faster than simplex), this benefit will translate to the efficiency of octatopes/hexatopes for LP-intensive applications like reachability analysis of neural networks. While the current state-of-the-art implementations of the algorithms for the MCF problem

are not as advanced as those for LP, we believe that this will change in light of the proposed application. We implement the octatope and hexatope abstract domains and show their effectiveness on several ACAS Xu networks [Kat+17], a popular benchmark for neural network verification.

Related Work

A growing body of research exists on different methods to verify neural networks [Liu+21], including recent tool competitions [BLJ21]. Algorithms can be categorized into search, optimization, and reachability solutions. In the space of search procedures, the seminal Reluplex method proposes an extension of the simplex algorithm used for linear programming to handle ReLU networks [Kat+17]. This method has been widely adopted and extended by, for example, posing verification as a constraint satisfaction problem [Kat+19]. This can then be solved using off-the-shelf Satisfiability-Modulo-Theory (SMT) solvers like z3 [MB08]. The use of SMT enables reasoning over different activation functions and topologies.

Interval arithmetic is another popular approach often used to estimate the range of output values given a range of inputs while tracking the input and output ranges of individual activation functions [Wan+18]. This can be computed by using linear programming to derive lower and upper bounds for a given node in the network. The work of [HL21] combines this with symbolic interval propagation and gradient descent to find counter-examples to the over-approximations established by the linear programming solutions. More sophisticated node splitting strategies that account for downstream effects on successor nodes can also be used as part of the symbolic interval propagation phase [HL20]. Per-neuron split constraints can also further improve efficiency [Wan+21].

Optimization solutions to the verification based on ILP have been explored. This is a natural formulation for the verification of neural networks due to the use of affine transformations and the fact that piecewise linear activation functions can be encoded using a set of binary linear constraints [Aki+18]. The work in [TXT19] extends similar ideas by estimating the maximum disturbance that is permitted at the input and proposing pre-solve procedures to speed up the solution.

Although solutions based on SMT-solving and mathematical programming are often complete,

they require the entire network to be encoded within the corresponding constraints, thereby limiting scalability. In contrast to these search and optimization solutions, the use of reachability analysis for verification of neural networks has been shown to scale to larger instances at the cost of completeness. Examples of this include the use of zonotope and star set abstract domains. The former can be efficiently employed to compute conservative over-approximations of output bounds of nodes in a network [GGP09], whereas linear programming can be employed for the latter to find tight bounds at the cost of scalability [Tra+20b]. The work proposed herein seeks to advance the state of verification methods based on reachability analysis by providing tighter over-approximations than zonotopes and more efficient computations than star sets.

A.1 Abstraction Based Neural Network Verification

We primarily work with feedforward neural networks (NNs) with rectified linear unit (ReLU) activation functions. We focus on networks with k fully-connected layers, also called multi-layer perceptrons. A ReLU is a commonly used activation function $\sigma : \mathbb{R} \to \mathbb{R}$ defined as $\sigma(x) = \max\{x, 0\}$. We can extend this function from scalars to vectors as $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ in a straightforward fashion by applying ReLU component-wise. This setup is the most typical situation considered for neural network verification tools [BLJ21], although extensions have been made to other layer types [Tra+20a; Tra+21] and activation functions [Sin+19].

Formally, a *neural network* is a function $f : \mathbb{R}^i \to \mathbb{R}^o$, where i is the input dimension and o is the output dimension. Each layer of the network is also a function $f_k : \mathbb{R}^{i_k} \to \mathbb{R}^{o_k}$ with its own input dimension i_k and output dimension o_k, defined as the composition $\sigma_k \circ \alpha_k$ in which $\alpha_k : \mathbb{R}^{i_k} \to \mathbb{R}^{o_k}$ is an affine mapping that is followed by the application of a ReLU $\sigma_k : \mathbb{R}^{o_k} \to \mathbb{R}^{o_k}$ of appropriate dimension. For the network to be well-formed, it is required that the input dimension of the first layer is i, the output dimension of the final layer is o, and all intermediate layers have input dimensions equal to the output dimension of the preceding layers and output dimension equal to the input dimension of the following layer. For a network with n layers, the function f is defined by the composition $f_n \circ f_{n-1} \circ ... \circ f_2 \circ f_1$.

Definition A.1 – Exact Range Computation Problem

Given a neural network implementing the function $f : \mathbb{R}^i \to \mathbb{R}^o$ and an input set $\mathcal{I} \subseteq \mathbb{R}^i$, the exact range computation problem is to compute the image

$$f(\mathcal{I}) = \{f(\boldsymbol{x}) : \boldsymbol{x} \in \mathcal{I}\}.$$

Definition A.2 – Neural Network Verification Problem

Given an input set $\mathcal{I} \subseteq \mathbb{R}^i$, an unsafe set $\mathcal{U} \subseteq \mathbb{R}^o$, and a neural network that computes $f : \mathbb{R}^i \to \mathbb{R}^o$, the neural network verification problem asks whether

$$f(\mathcal{I}) \cap \mathcal{U} = \emptyset.$$

As is typical with the state-of-practice in neural network verification, we restrict the input sets and unsafe sets to be defined by systems of linear constraints

$$\mathcal{I} = \left\{\boldsymbol{x} \in \mathbb{R}^i : \boldsymbol{M}_i \boldsymbol{x} \leq b_i\right\}, \qquad \text{and} \qquad \mathcal{U} = \{\boldsymbol{x} \in \mathbb{R}^o : \boldsymbol{M}_u \boldsymbol{x} \leq b_u\}.$$

A.1.1 Abstraction-Based Approach

We now describe a general approach to the neural network verification problem that is based on interpreting sets in the domains and codomains of NNs (and their component layers) with respect to some *abstract domain* which, in our case, comprises a particular class of geometric objects. Suppose, for the time being, that an appropriate abstract domain has already been chosen. Given an instance of the neural network verification problem, the abstraction based paradigm proceeds roughly as presented in Algorithm 8.

Algorithm 8: Abstract Neural Network Verification.

Data:
- $f_k = \sigma_k \circ \alpha_k$, for each layer k in the given NN;
- input set $\mathcal{I}$;
- unsafe set $\mathcal{U}$;

```
Identify elements A, B of the abstract domain such that I ⊆ A and U ⊆ B;
for each layer k do
    Compute the image f_k(A);
    Find an element A' of the abstract domain such that f_k(A) ⊆ A';
    Set A ← A';
return A ∩ B = ∅;
```

From the above algorithm, we distill three operations that are fundamental to the overall approach:

- *affine transformation* is required to compute $\alpha_k(\mathcal{A})$;
- *half-space intersection* is necessary for computing $\sigma_k(\alpha_k(\mathcal{A}))$;
- *emptiness determination* is essential for deciding the final return value.

For the purposes of abstract neural network verification, any reasonable abstract domain should facilitate algorithmic execution of these operations.

Besides the feasibility of carrying out the critical operations, additional properties of an abstract domain that are helpful in this context may be inferred from Algorithm 8. If a given abstract domain is *closed under affine transformation*, then the instruction on line 5 may be omitted. If the abstract domain is *closed under intersection*, the emptiness check to determine the return value is simplified.

We proceed by examining two particular geometric abstract domains that are useful in the context of abstract neural network verification.

A.1.2 Zonotopes

A relatively simple abstract domain is the family of *zonotopes*.

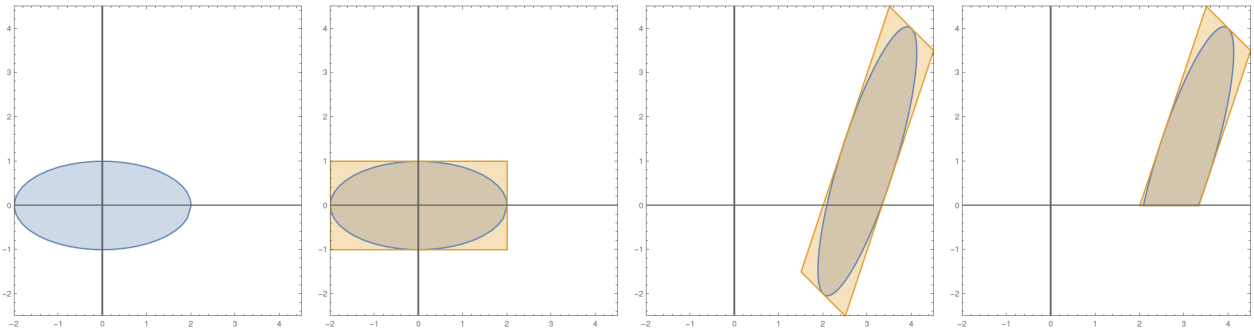
Definition A.3 – Zonotope

An n-dimensional zonotope is the image of a p-dimensional hypercube under an affine transformation $\mathbb{R}^p \to \mathbb{R}^n$. It is given as a pair $Z = \langle \boldsymbol{c}, \boldsymbol{G} \rangle$ comprising a center $\boldsymbol{c} \in \mathbb{R}^n$ and a collection of generator vectors $\boldsymbol{g}_1, ..., \boldsymbol{g}_p \in \mathbb{R}^n$ forming a matrix $\boldsymbol{G} = \begin{bmatrix} \boldsymbol{g}_1 & \cdots & \boldsymbol{g}_p \end{bmatrix} \in \mathbb{R}^{n \times p}$. The semantics of Z are defined as

$$[\![Z]\!] = \left\{ \boldsymbol{G}\boldsymbol{x} + \boldsymbol{c} : -\mathbf{1}_p \leq \boldsymbol{x} \leq \mathbf{1}_p \right\}.$$

Now that we have concretely defined a particular abstract domain, the process of Algorithm 8 can be explicitly visualized (in the two-dimensional case). Figure A.1 depicts the first iteration of the algorithm using zonotopes as the abstract domain:

- Figure A.1a displays the initial set, which is an elliptical region in the plane in this case;
- Figure A.1b overlays the initial set with a box;



(a) Initial region (b) Over approximation (c) Affine transformation (d) ReLU

Figure A.1: An illustration of the first step of Algorithm 8 using the zonotope abstract domain.

- Figure A.1c shows an affine image of the initial set and its approximating box;
- Figure A.1d plots the result of applying a ReLU after the affine mapping.

A.1.3 Linear Star Sets

A more expressive abstract domain is the family of *linear star sets*.

Definition A.4 – Linear Star Set

Linear star sets generalize zonotopes by letting the kernel be defined by an LCS; a linear star set is the image of a p-dimensional polytope under an affine transformation $\mathbb{R}^p \to \mathbb{R}^n$. Formally, an n-dimensional star set S is specified as a tuple $\langle \boldsymbol{c}, \boldsymbol{G}, \boldsymbol{M}, \boldsymbol{b} \rangle$ including a matrix $\boldsymbol{M}$ and vector $\boldsymbol{b}$ representing the polytope $\boldsymbol{M}\boldsymbol{x} \leq b$, a center $\boldsymbol{c} \in \mathbb{R}^n$, and a collection of generator vectors $\boldsymbol{g}_1, ..., \boldsymbol{g}_p \in \mathbb{R}^n$ that form a matrix $\boldsymbol{G} = \begin{bmatrix} \boldsymbol{g}_1 & \cdots & \boldsymbol{g}_p \end{bmatrix} \in \mathbb{R}^{n \times p}$. The semantics of S are defined as

$$[\![S]\!] = \{\boldsymbol{G}\boldsymbol{x} + \boldsymbol{c} : \boldsymbol{M}\boldsymbol{x} \leq \boldsymbol{b}\}.$$

A.1.4 Kernels Of Abstract Domains

Both a zonotope and a linear star set may be viewed as an n-dimensional image of a polytope—which we refer to as the kernel—under affine transformation. For zonotopes, the kernel is a hypercube, while for linear star sets the kernel is a set defined by a generic polytope. Thus, the kernel of a linear star set may be specified as a linear constraint system. The kernel of a zonotope can be encoded by an *interval constraint system*, which is a system of linear constraints of the form $a_i \leq x_i \leq b_i$ where

$a_i, b_i \in \mathbb{R}$. In the present work, we examine geometric abstract domains that fall between zonotopes and linear star sets in terms of precision and complexity. Such domains shall be characterized by the subclasses of linear constraint systems that specify their kernels.

We now state formally the results [Tra+19] about linear star sets that are fundamental to our approach towards neural network verification. Each of the following theorems is key to ensuring that the operations of affine transformation, half-space intersection, and emptiness checking are available and feasible for linear star sets. Since each abstract domain considered in this paper forms a subclass of linear star sets, these serve as a template for the stronger results we subsequently establish for hexatopes and octatopes.

Theorem A.1

Linear star sets are closed under affine transformation.

Theorem A.2

The problems of linear optimization and emptiness checking over linear star sets can be solved in polynomial time by reduction to linear programming.

Theorem A.3

The intersection of a linear star set $S = \langle \boldsymbol{c}, \boldsymbol{G}, \boldsymbol{M}, \boldsymbol{b} \rangle$ and half space $\{\boldsymbol{y} \mid \boldsymbol{H}\boldsymbol{y} \leq \boldsymbol{d}\}$ is another linear star set $S' = \langle \boldsymbol{c}, \boldsymbol{G}, \boldsymbol{M}', \boldsymbol{b}' \rangle$ where $\boldsymbol{M}'\boldsymbol{x} \leq \boldsymbol{b}'$ comprises the conjunction of constraints

$$\boldsymbol{M}\boldsymbol{x} \leq \boldsymbol{b} \qquad \text{and} \qquad \boldsymbol{M}\boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{d} - \boldsymbol{H}\boldsymbol{c}.$$

In the sequel, we generalize the notion of zonotopes to define *hexatopes* and *octatopes*, two intermediate abstract domains that fall between zonotopes and linear star sets in terms of expressive power. We develop a series of results, analogous to Theorems A.1 to A.3, that provide the theoretical foundation to enable the application of these abstract domains to neural network verification.

A.2 Hexatopes

Definition A.5 – Difference Constraint System

A difference constraint is a linear constraint of the form

$$x_i - x_j \leq b, \qquad \text{where} \qquad b \in \mathbb{R}.$$

A difference constraint system (DCS) is a conjunction of difference constraints.

Definition A.6 – Hexatope

A hexatope $\boldsymbol{H} = \langle \boldsymbol{c}, \boldsymbol{G}, \boldsymbol{A}, \boldsymbol{b} \rangle$ is a special type of linear star set, having a kernel $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ defined by a difference constraint system.

Our first result mirrors Theorem A.1 and establishes closure under affine mappings for hexatopes.

Theorem A.4

Hexatopes are closed under affine transformation.

Proof. From Theorem A.1 it follows for any hexatope $\boldsymbol{H} = \langle \boldsymbol{c}, \boldsymbol{G}, \boldsymbol{A}, \boldsymbol{b} \rangle$ and any affine mapping $f(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{d}$, that $\{\boldsymbol{W}\boldsymbol{x} + \boldsymbol{d} : \boldsymbol{x} \in [\![\boldsymbol{H}]\!]\}$ is linear star set $\boldsymbol{H}' = \langle \boldsymbol{c}', \boldsymbol{G}', \boldsymbol{A}, \boldsymbol{b} \rangle$ where

$$\boldsymbol{c}' = \boldsymbol{W}\boldsymbol{c} + \boldsymbol{d} \qquad \text{and} \qquad \boldsymbol{G}' = \begin{bmatrix} \boldsymbol{W}\boldsymbol{g}_1 & \cdots & \boldsymbol{W}\boldsymbol{g}_p \end{bmatrix}.$$

Since the transformation does not change the kernel, $\boldsymbol{H}'$ is indeed a hexatope. □

A.2.1 Linear Optimization via Minimum Cost Flow

In this subsection, we show that for any linear optimization problem over a difference constraint system, the dual problem can be reduced to the minimum cost flow problem, which we now review.

Definition A.7

A *flow network* $\boldsymbol{G} = (V, E, c, a, d)$ is a directed graph $\boldsymbol{G} = (V, E)$ with a capacity $c : E \to \mathbb{R}_{\geq 0}$ and a cost $a : E \to \mathbb{R}$ associated with every edge (arc) and a demand $d : V \to \mathbb{R}$ associated with every vertex (node). We assume that $\sum_{v \in V} d(v) = 0$.

Algorithm 9: Out-of-Kilter

Input: Flow network $\boldsymbol{G} = (V, E, c, a, d)$

Initialize the potential as 0 for every node;
Let f be a flow in $\boldsymbol{G}$;
Construct the residual network $\boldsymbol{G}_f$;
Compute the kilter number $k(u,v)$ of each edge (u,v) in $\boldsymbol{G}_f$;
while $\boldsymbol{G}_f$ *contains an edge with positive kilter number* **do**
    Select an edge (u,v) in $\boldsymbol{G}_f$ with positive kilter number;
    Let the weight of each edge (u,v) in $\boldsymbol{G}_f$ be $\max\{0, c^\pi(u,v)\}$;
    Let P be a shortest path from v to u;
    Let $l(w)$, for each $w \in V \setminus \{u,v\}$, be the weight of the least weight path from v to w;
    For each node w, set $\pi(w) \leftarrow \pi(w) - l(w)$;
    if $c^\pi(u,v) < 0$ **then**
        Set $Q \leftarrow P \cup \{(u,v)\}$;
        $\delta \leftarrow \min_{(u,v)\in Q} r(u,v)$;
        Augment δ units of flow along Q;
        Update f and $\boldsymbol{G}_f$;
return f;

The *minimum cost flow* (MCF) problem can be stated as follows:

$$\begin{aligned} \text{Minimize} \quad & \sum_{(u,v)\in E} f(u,v)a(u,v) & \\ \text{subject to} \quad & \sum_{u\in V} f(u,v) - \sum_{u\in V} f(v,u) = d(v) & \text{for all } v \in V, \\ & 0 \le f(u,v) \le c(u,v) & \text{for all } (u,v) \in E. \end{aligned}$$

It is well known that there exist strongly polynomial time algorithms [GT88; GT89] for the MCF problem. One example is the Out-of-Kilter algorithm, which also produces a solution to the dual problem [AMO93], that we now review.

Pseudocode for the Out-of-Kilter algorithm is given in Algorithm 9. It starts with a possibly infeasible flow and iteratively modifies this flow in a way that decreases the infeasibility of the solution and moves it closer to optimality. Each step of the algorithm consists of solving a shortest path problem and augmenting the flow along the shortest path. It operates on the residual network $\boldsymbol{G}_f$ corresponding to the current flow f. This residual network is constructed as follows.

Feasible Edges: If $f(u,v) < c(u,v)$, we add the edge (u,v) with a residual capacity of $r(u,v) = c(u,v) - f(u,v)$ and cost $a(u,v)$. If $f(u,v) > 0$, we add the edge (v,u) with a residual capacity of

$r(v,u) = f(u,v)$ and cost $-a(u,v)$.

Lower-Infeasible Edges: If $f(u,v) < 0$, we add edge (u,v) with residual capacity $r(u,v) = -f(u,v)$ and cost $a(u,v)$.

Upper-Infeasible Edges: If $f(u,v) > c(u,v)$, we add the edge (v,u) with residual capacity $r(v,u) = f(u,v) - c(u,v)$ and cost $-a(u,v)$.

For each vertex v in the residual network, the algorithm maintains a potential $\pi(v)$ and for each edge (u,v) with cost $a(u,v)$, it maintains the reduced cost $a^\pi(u,v) = c(u,v) - \pi(u) + \pi(v)$. Additionally, for each edge in the residual network, it maintains a kilter number $k(u,v)$ which is 0 if $c^\pi(u,v) \geq 0$ and is the residual capacity $r(u,v)$ if $c^\pi(u,v) < 0$. This kilter number represents the change in flow required so that each edge satisfies its optimality condition.

Note that the node potentials π and reduced costs c^π corresponding to the optimal flow f are the optimal solution of the dual problem [AMO93]. The Out-of-Kilter algorithm runs in time $O\big(D\big(m^2 + mn\log n\big)\big)$ on a network with m edges and n vertices and maximum demand D.

Theorem A.5

The linear optimization problem over hexatopes can be solved in strongly polynomial time via reduction to the minimum cost flow problem.

Proof. Consider an n-dimensional hexatope $\boldsymbol{H} = \langle \boldsymbol{c}, \boldsymbol{G}, \boldsymbol{A}, \boldsymbol{b} \rangle$ which is the image of a p-dimensional DCS-defined set. In order to optimize a linear function f over $[\![\boldsymbol{H}]\!]$, it suffices to optimize the composition of functions $f \circ h$ where $h(\boldsymbol{x}) = \boldsymbol{G}\boldsymbol{x} + \boldsymbol{c}$ over the difference constrained set $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$. Suppose that $f(\boldsymbol{x}) = \boldsymbol{f}\boldsymbol{x} = \sum_k f_k x_k$, then the composite objective function can be written as $f \circ h(\boldsymbol{x}) = \boldsymbol{f}\boldsymbol{G}\boldsymbol{x} + \boldsymbol{f}\boldsymbol{c}$. Then, omitting the constant term $\boldsymbol{f}\boldsymbol{c}$ which can be reincorporated in the end, we have that $f \circ h(\boldsymbol{x}) = \sum_k w_k x_k$ for some appropriate coefficients w_k.

Following section 4 of [Cor+09a], we construct a directed graph, called the constraint graph, to represent the difference constraint system determining the kernel of the given hexatope. For each variable x_i, there is a vertex v_i in the constraint graph. Additionally, there is one extra vertex v_0. Set the demand of each vertex as $d(v_i) = w_i$, for all $i > 0$, and let $d(v_0) = -\sum_k w_k$. For every constraint of

the form $x_i - x_j \leq b$ in the DCS, there is an edge (v_j, v_i) in the constraint graph with cost b and infinite capacity. For every vertex v_i, with $i > 0$, there is also an edge (v_0, v_i) with cost 0 and infinite capacity.

The MCF problem instance constructed in this manner is equivalent to the dual of the given linear optimization problem instance over the given DCS. Since the Out-of-Kilter algorithm also solves the dual to the minimum cost flow problem [AMO93], running it on the dual of the DCS optimization problem will also solve the DCS optimization problem itself. For a DCS with m constraints, this process takes $O((m^2 + mp\log p)C)$ time, where C is the largest absolute value of any coefficient in the objective function. □

A.3 Octatopes

Definition A.8 – UTVPI Constraint System

A Unit-Two-Variable-Per-Inequality (UTVPI) constraint is a linear constraint of the form

$$a_i x_i + a_j x_j \leq b, \qquad \text{where} \qquad a_i, a_j \in \{-1, 0, 1\} \qquad \text{and} \qquad b \in \mathbb{R}.$$

A UTVPI constraint system (UCS) is a conjunction of UTVPI constraints.

A UTVPI constraint $a_i x_i + a_j x_j \leq b$ is said to be an absolute constraint if $a_i = 0$ or $a_j = 0$. An absolute constraint can be converted into constraints of the form: $a_i x_i + a_j x_j \leq b$, where both a_i and a_j are non-zero. Note that a UTVPI constraint $a_i x_i + a_j x_j \leq b$ is a difference constraint if $a_i = -a_j$. The constant that bounds a UTVPI constraint is called the defining constant. For instance, the defining constant for the constraint $x_1 - x_2 \leq 9$ is 9.

Definition A.9 – Octatope

An octatope is a special kind of linear star set $\langle \boldsymbol{c}, \boldsymbol{G}, \boldsymbol{A}, \boldsymbol{b} \rangle$, having a kernel $\boldsymbol{Ax} \leq \boldsymbol{b}$ defined by a UTVPI constraint system.

Following the same argument used to prove Theorem A.1, we establish closure of octatopes under affine mappings.

Theorem A.6

Octatopes are closed under affine transformation.

By Theorem A.2, linear optimization over linear star sets can be done in polynomial time. Our next result shows that linear optimization over octatopes and hexatopes can be done in *strongly* polynomial time.

Theorem A.7

The linear optimization problem for octatopes can be solved in strongly polynomial time via reduction to the linear optimization problem for hexatopes.

Proof. Following techniques of [LM05; Min01; Min06], we convert a UCS $\boldsymbol{U}$ into a DCS $\boldsymbol{D}$. The first part of the conversion creates the variables x_i^+ and x_i^- in $\boldsymbol{D}$ for each variable x_i in $\boldsymbol{U}$. Then, each constraint in $\boldsymbol{U}$ is converted as follows:

(1) Each constraint of the form $x_i + x_j \leq b$ becomes two constraints

$$x_i^+ - x_j^- \leq b \qquad \text{and} \qquad -x_i^- + x_j^+ \leq b.$$

(2) Each constraint of the form $x_i - x_j \leq b$ becomes two constraints

$$x_i^+ - x_j^+ \leq b \qquad \text{and} \qquad -x_i^- + x_j^- \leq b.$$

(3) Each constraint of the form $-x_i + x_j \leq b$ becomes two constraints

$$x_i^- - x_j^- \leq b \qquad \text{and} \qquad -x_i^+ + x_j^+ \leq b.$$

(4) Each constraint of the form $-x_i - x_j \leq b$ becomes two constraints

$$x_i^- - x_j^+ \leq b \qquad \text{and} \qquad -x_i^+ + x_j^- \leq b.$$

(5) Each constraint of the form $x_i \leq b$ becomes a constraint

$$x_i^+ - x_i^- \leq 2b.$$

(6) Each constraint of the form $-x_i \leq b$ becomes constraint

$$x_i^- - x_i^+ \leq 2b.$$

Observe that $x_i = \frac{1}{2}(x_i^+ - x_i^-)$ satisfies the original UCS. Thus, we can consider this as the problem maximizing the objective function over variables $\frac{1}{2}(x_i^+ - x_i^-)$ of the DCS $\boldsymbol{D}$. □

A.3.1 Emptiness Checking

We also consider the feasibility problem for octatopes. That is, the problem of deciding whether an octatope is empty.

Theorem A.8

The emptiness of an octatope can be decided in $O(pm)$ time and $O(p+m)$ space where p is the number of generator vectors and m is the number of UTVPI constraints defining its kernel.

Proof. It is easy to see that an octatope is empty if and only if the UTVPI constraints of its kernel are unsatisfiable as linear mappings over polytopes that are monotone with respect to set inclusion. The complexity then follows from results on checking the feasibility of UTVPI constraint systems [LM05]. □

A.3.2 Half-Space Intersection

It follows from Theorem A.3 that the intersection of an octatope $O = \langle \boldsymbol{c}, \boldsymbol{G}, \boldsymbol{A}, \boldsymbol{b} \rangle$ and half space $\{\boldsymbol{y} \mid \boldsymbol{M}\boldsymbol{y} \leq \boldsymbol{d}\}$ is a star set $O' = \langle \boldsymbol{c}, \boldsymbol{G}, \boldsymbol{A}', \boldsymbol{b}' \rangle$ where the constraints $\boldsymbol{A}'\boldsymbol{x} \leq \boldsymbol{b}'$ are the conjunction of UCS constraints $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ and the hyperplane $\boldsymbol{M}\boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{d} - \boldsymbol{M}\boldsymbol{c}$. In the rest of this section, we show how an over-approximation of this intersection can be represented as UCS constraints. The treatment for hexatopes is similar, and hence omitted.

We formalize this problem as the *UTVPI bounding box problem*.

Definition A.10 – UTVPI Bounding Box

Given a UCS $\boldsymbol{U}$ and an arbitrary linear constraint l, a *UTVPI bounding box* is a UCS $\boldsymbol{U}'$, such that every solution to $\boldsymbol{U} \cup \{l\}$ is a solution to $\boldsymbol{U}'$. For a given UCS $\boldsymbol{U}$ and constraint l, a *tightest* UTVPI bounding box is a bounding box of $\boldsymbol{U} \cup \{l\}$ that is contained within every other bounding box of $\boldsymbol{U} \cup \{l\}$.

Algorithm 10: UTVPIBoundingBox($\boldsymbol{U}, l$)

Input: UCS $\boldsymbol{U}$ and constraint l
Output: A UTVPI bounding box $\boldsymbol{U}'$

Initialize $\boldsymbol{U}'$ to $\emptyset$;
forall *pairs of variables* x_i, x_j *in* $\boldsymbol{U}$ **do**
  Let $u_{ij}^{+-} = \max_{\boldsymbol{U}\cup\{l\}} x_i - x_j$ and add constraint $x_i - x_j \leq u_{ij}^{+-}$ to $\boldsymbol{U}'$;
  Let $u_{ij}^{-+} = \max_{\boldsymbol{U}\cup\{l\}} x_j - x_i$ and add constraint $x_j - x_i \leq u_{ij}^{-+}$ to $\boldsymbol{U}'$;
  Let $u_{ij}^{++} = \max_{\boldsymbol{U}\cup\{l\}} x_i + x_j$ and add constraint $x_i + x_j \leq u_{ij}^{++}$ to $\boldsymbol{U}'$;
  Let $u_{ij}^{--} = \max_{\boldsymbol{U}\cup\{l\}} -x_i - x_j$ and add $-x_i - x_j \leq u_{ij}^{--}$ to $\boldsymbol{U}'$;
  Let $u_i^{+} = \max_{\boldsymbol{U}\cup\{l\}} x_i$ and add constraint $x_i \leq u_i^{+}$ to $\boldsymbol{U}'$;
  Let $u_i^{-} = \max_{\boldsymbol{U}\cup\{l\}} -x_i$ and add constraint $-x_i \leq u_i^{-}$ to $\boldsymbol{U}'$;
return $\boldsymbol{U}'$;

Thus, a UTVPI bounding box of a UCS $\boldsymbol{U}$ and constraint l is a UCS that overestimates the solution space of $\boldsymbol{U} \cup \{l\}$. A tightest bounding box is a UCS that overestimates the solution space the least. Each of the linear programs used to construct $\boldsymbol{U}'$ can be solved (with L bits of precision) in $O(n^{2.38}L)$ time [CLS19; CLS21]. Since finding the UTVPI bounding box requires solving $O(n^2)$ linear programs, the UTVPI bounding box can be found in $O(n^{4.38}L)$ time.

Theorem A.9

Let $\boldsymbol{U}$ be a UCS and let l be an arbitrary linear constraint. The UCS $\boldsymbol{U}'$, constructed by Algorithm 10, is a UTVPI bounding box of $\boldsymbol{U} \cup \{l\}$.

Proof. Let $\boldsymbol{x}^*$ be a solution to $\boldsymbol{U}\cup\{l\}$. Let $a_i x_i + a_j x_j \leq u_{ij}$ be an arbitrary constraint in $\boldsymbol{U}'$. By construction of $\boldsymbol{U}'$, we have $u_{ij} = \max_{\boldsymbol{U}\cup\{l\}} a_i x_i + a_j x_j$.

Since $\boldsymbol{x}^*$ is a solution of $\boldsymbol{U} \cup \{l\}$, $a_i x_i^* + a_j x_j^* \leq u_{ij}$. This means that $\boldsymbol{x}^*$ satisfies the constraint $a_i x_i + a_j x_j \leq u_{ij}$. Since the constraint $a_i x_i + a_j x_j \leq u_{ij}$ was chosen arbitrarily, $\boldsymbol{x}^*$ is a solution to $\boldsymbol{U}'$. Note that $\boldsymbol{x}^*$ was an arbitrary solution to $\boldsymbol{U} \cup \{l\}$. Thus, every solution to $\boldsymbol{U} \cup \{l\}$ is a solution to $\boldsymbol{U}'$. Consequently, $\boldsymbol{U}'$ is a UTVPI bounding box of $\boldsymbol{U} \cup \{l\}$. □

We now show that $\boldsymbol{U}'$ is a tightest UTVPI bounding box of $\boldsymbol{U} \cup \{l\}$. Note that $\boldsymbol{U} \cup \{l\}$ must have a tightest bounding box. Consider two bounding boxes $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ of $\boldsymbol{U} \cup \{l\}$. Let $\boldsymbol{U}^*$, be the UCS formed by combining the constraints in $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$. Note that $\boldsymbol{U}^*$ is also a bounding box of $\boldsymbol{U} \cup \{l\}$. Additionally, every solution to $\boldsymbol{U}^*$ is a solution to both $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$. Thus, if $\boldsymbol{U} \cup \{l\}$ has two incomparable bounding

boxes, then a new bounding box can be constructed that is tighter than both.

Theorem A.10

Let $\boldsymbol{U}$ be a UCS and let l be a linear constraint. The UCS $\boldsymbol{U'}$, produced by Algorithm 10, is a tightest UTVPI bounding box of $\boldsymbol{U} \cup \{l\}$.

Proof. Assume for the sake of contradiction, that $\boldsymbol{U'}$ is not a tightest UTVPI bounding box of $\boldsymbol{U} \cup \{l\}$. Thus, there exist a UTVPI bounding box $\boldsymbol{U''}$ and a point $\boldsymbol{x^*}$ such that $\boldsymbol{x}^*$ is a solution to $\boldsymbol{U'}$, but not a solution to $\boldsymbol{U''}$. This means that there is a UTVPI constraint $a_i x_i + a_j x_j \leq b$ in $\boldsymbol{U''}$ that is violated by $\boldsymbol{x^*}$.

Let $u_{ij} = \max_{\boldsymbol{U} \cup \{l\}} a_i x_i + a_j x_j$. Since $\boldsymbol{U''}$ is a UTVPI bounding box of $\boldsymbol{U} \cup \{l\}$, every solution to $\boldsymbol{U} \cup \{l\}$ is a solution to $\boldsymbol{U''}$. Thus, every solution to $\boldsymbol{U} \cup \{l\}$ satisfies the constraint $a_i x_i + a_j x_j \leq b$. This means that

$$\max_{\boldsymbol{U} \cup \{l\}} a_i x_i + a_j x_j$$

is bounded from above by b. Thus, u_{ij} exists and $u_{ij} \leq b$.

By the construction of $\boldsymbol{U'}$, the constraint $a_i x_i + a_j x_j \leq u_{ij}$ is in $\boldsymbol{U'}$. However, $\boldsymbol{x^*}$ is a solution to $\boldsymbol{U'}$ such that $a_i x_i^* + a_j x_j^* > b \geq u_{ij}$. This is a contradiction. Thus, $\boldsymbol{U'}$ must be a tightest UTVPI bounding box of $\boldsymbol{U} \cup \{l\}$. □

A.4 Experimental Results

The exact range computation problem from Definition A.1 can be solved using linear star sets (see Algorithms 1 and 2 in earlier work for a full review [Bak+20]).

The neural network function f as defined in Appendix A.1 is a piece-wise affine function of the inputs. The range computation proceeds using geometric set operations. The initial set of states is represented as a linear star set and propagated through each layer of the network. To go from the output of one layer to the vector of intermediate values at the next layer, an affine transformation operation is performed on the set. The effect of the ReLU activation in a layer is handled iteratively for each neuron. The set of states is potentially split along the neuron input constraint $y_i = 0$, into a negative region and a positive region, using a half-space intersection operation. The negative region is then projected to zero

to match the semantics of a ReLU. The two sets are then considered independently for the remaining neurons in the layer, as well as the rest of the layers in the network. For a given input set, not all neurons require splitting the set in two, since the input constraints may restrict inputs to be strictly positive or negative. To check this, before splitting we first optimize over the set in the direction of the intermediate value $x_j^{(i)}$ corresponding to a specific neuron j in layer i. If splitting occurs, the two sets are treated independently and propagated through the remaining neurons in the layer, possibly requiring further splitting in the remaining parts of the network.

After applying a number of optimizations, the bottleneck of exact range computation with star sets is the use of LP solving to compute the input bounds for each neuron [Bak+20]. To improve analysis speed, rather than speeding up LP solving—which is a well-studied problem where further progress is likely to be difficult—we instead seek methods that can reduce the number of LPs needed.

In earlier work, zonotope abstract domains have been considered for this task. Rather than just propagating star sets through a network, we also propagate a zonotope overapproximation that we use in a *prefiltering* step. Recall that before splitting we first need to optimize over the set in the direction of the intermediate value $x_j^{(i)}$. Before optimizing over the star set using LP, we first optimize over the zonotope abstraction prefilter. If the zonotope abstraction can prove that the inputs are strictly positive or negative, then we are guaranteed the exact result from the LP will be strictly positive or negative as well (as the zonotope is an overapproximation of the star set). This allows us to avoid LP, as optimization over zonotopes can be done efficiently using a simple loop.

The reason zonotope analysis is not exact is that zonotopes do not support general half-space intersections when sets must be split. Instead, two approaches have been considered. The easiest option is to ignore intersections, which is fast but can cause significant overapproximation error in the abstraction [Geh+18; Tra+21]. Alternatively, we can perform *domain contraction*, which is to search for zonotopes that more tightly overapproximate the intersection. Different approaches for domain contraction are possible, ranging from reasoning methods over individual constraints to more accurate approaches that use LP solving on the star set in the generator coefficient space [Bak+20]. Although the LP approach uses the expensive operation we are trying to reduce, it can result in an overall reduction of

LPs, as the neuron input bounds can be computed more accurately.

This work proposes using octatope abstract domains as a prefilter. As described earlier, optimization over octatopes can be done more efficiently than general LP solving. The greater expressiveness of octatopes compared with zonotopes means that we can hope to further reduce the number of LPs needed with the star set when computing a neuron's input bounds for splitting. We evaluate this impact in our experiments. In terms of handling intersections when splitting sets, octatopes (like zonotopes) cannot exactly support any general half-space intersection operation. This means that a domain contraction step may be necessary to ensure tight overapproximation.

We next evaluate the potential savings in LP computation to computing neuron input bounds during exact range computation for neural networks. Our evaluation is performed on several benchmarks from the ACAS Xu benchmark suite [Kat+17], specifically focusing on property 3 and 4 where earlier work has shown exact range computation is tractable [Bak+20]. We generally report number of LPs for different operations rather than runtime, as the runtime is influenced by other factors such code optimizations and the choice of LP solver.

First, we examine the number of LPs needed to perform neuron input range computation, for different choices of prefilter abstract domain. The LP calls to find the neuron input ranges is the bottleneck of the overall range computation algorithm, so its reduction is of particular importance. The results are in Table A.1. The Star-Only approach uses only LP solving with no prefilter, and therefore has the highest number of LPs. The next column, Zonotope-NC corresponds to the case where zonotope prefilters are used, but no domain contraction is performed (half-space intersections are ignored). This has a significant reduction on the number of LP calls, for example in the first row with property 3 and network 1-6, where the number of LP calls is reduced from 91K to 11K. Using domain contraction with zonotopes, Zono-C, further reduces this to around 3.3K. The more precise domains with hexatopes and octatopes can further reduce this to around 2.6K and 2.5K, respectively. The minimum column is computed by seeing how many bounds computations could not be eliminated as they correspond to cases where the input to a neuron truly can be either positive or negative. Even a perfect prefilter could not eliminate these LPs, as prefilters only eliminate cases where splitting is impossible. Other

Prop	Net	Star-Only	Zono-NC	Zono-C	Hex	Oct	Minimum
3	1-6	91762	11152	3382	2635	2571	1886
3	2-7	77896	9365	2921	2240	2198	1626
3	3-5	80988	8990	2711	2131	2092	1710
3	5-2	54758	15523	7762	6820	6704	3779
4	1-4	53036	7736	2597	2389	2330	1926
4	2-7	38748	3851	1249	888	861	753
4	5-9	68750	8814	2952	2286	2151	1591

Table A.1: Number of LP calls to find neuron input bounds for different abstract domain prefilters on various ACASXu properties and networks.

approaches could be considered to remove these LPs, such as tracking specific witness input points that can prove a neuron can have both positive and negative inputs, which we may consider in future work. Overall, the proposed octatope abstract domain has the potential to reduce the number of unnecessary LPs significantly in exact range computation.

When using the new abstract domain, however, there is a trade-off where extra operations are needed to perform domain contraction as well as to optimize within the abstract domains. We used a witness-tracking approach [Bak21], where for each constraint a witness point was included that was in the star set and on the boundary of the constraint. When new intersections are performed, each witness point is checked to see if it is now excluded from the set. When points are excluded, new witness points get generated by solving an LP in the direction of the constraint, which may tighten the constraint. This results in the tight abstract domains, but can be expensive when many constraints are possible. For hexatopes and octatopes, the number of possible constraints is quadratic in the number of variables (ACAS Xu has 5 input variables).

Table A.2 shows the number of LPs needed for each example when performing domain contraction. Star-Only and Zono-NC do not perform domain contraction, and so have 0 LPs for this operation. As expected, the more complex the abstract domain, the more operations are needed. This is due to the contraction method performed, where the number of possible LPs needed at a domain contraction step increases as the number of possible constraints increases.

In terms of the performance of network simplex for optimizing within the octatope domain, the

Prop	Net	Star-Only	Zono-NC	Zono-C	Hex	Oct
3	1-6	0	0	12765	38400	115200
3	2-7	0	0	12280	36840	110520
3	3-5	0	0	10407	31230	93690
3	5-2	0	0	21249	63750	191250
4	1-4	0	0	11828	35493	106476
4	2-7	0	0	5533	16620	49860
4	5-9	0	0	9906	29730	89190

Table A.2: Number of LP calls for the domain contraction step for different abstract domain prefilters for various ACAS Xu properties and networks.

engineering aspect of the problem also requires further development. When computing the range of network 2-7 with the input set from property 4, the UTVPI constraints were optimized 38748 times. When using the commercial LP solver Gurobi on these constraints, each call took on average of 0.17ms. Formulating the min-cost flow problem and calling the `network_simplex` implementation from the `networkx` python library, however, used about 1.9 ms per call, about 11x slower. Further, while Gurobi always obtained a result, numerical issues caused network simplex to fail about 0.65% of the time.

In summary, while octatopes effectively reduce the bottleneck step of input bounds computation, further improvements must be made to octatope domain contraction algorithms as well as to implementation optimizations of min-cost flow solvers, before an overall speedup can be achieved. Nonetheless, it is an encouraging result for neural network verification as developing more efficient domain contraction algorithms and improving min-cost flow implementations is likely easier than coming up with new ways to speed up LP solving.

Index