NONSTANDARD MODELS OF THE WEAK SECOND ORDER THEORY OF ONE SUCCESSOR

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Thomas Robert Kern

May 2016

NONSTANDARD MODELS OF THE WEAK SECOND ORDER THEORY OF ONE SUCCESSOR

Thomas Robert Kern, Ph.D.

Cornell University 2016

In this paper, we seek to classify the nonstandard models of the weak (monadic) second order theory of one successor (WS1S), the theory of the two-typed structure consisting of natural numbers and finite sets of natural number equipped with a successor ($+1$) operation, and membership relation between the two types. This will require an automata-theoretic approach. Chapter 1 will provide an introductory background to the intersection of automata theory and model theory. In chapter 2, we use the Krohn-Rhodes Theorem and an observation about the powerset determinization construction to prove what will be essentially be a quantifier-elimination type result for our theory, although since the result is of independent interest, it will be presented in a more general automata-theoretic context as a generating set for the regular functions under composition. In chapter 3, we provide an axiomatization for WS1S. In chapter 4, we apply this axiomatization to prove a number of key theorems regarding nonstandard models of WS1S. This includes a classification of the possible first order types, tools for completely classifying nonstandard models, an exploration of countable nonstandard models with the simplest nontrivial first order type, and a tool for producing new nonstandard models given old nonstandard models.

BIOGRAPHICAL SKETCH

Thomas Kern received a Bachelor of Arts from Dartmouth College in 2009. There he served in various leadership positions for the Dartmouth Math Society. He participated in the Budapest Semesters in Mathematics program in Fall of 2007. This thesis represents his efforts towards obtaining a Ph.D. in Mathematics from Cornell University in May 2016. He will additionally be receiving a Masters in Computer Science from Cornell University in May 2016.

To those who helped me along my journey,

and to those who kept me company along the way.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grants No. DMS-0852811 and DMS-1161175. This work would not have been possible without encouragement and guidance from my advisor, Anil Nerode. I would also like to thank Scott Messick for valuable discussions on the theory of finite automata.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this series of papers, we utilize the Büchi Theorem, which formalizes the expressive equivalence between finite state automata and weak monadic second order logic, to explore model theoretic properties of the weak (monadic[1]) second order theory of one successor. Specifically, we will try to answer a number of questions about the nonstandard models of this theory.

The study of finite automata began in the 1950s with Stephen Kleene's work on Nerve Nets[6], an early form of neural network. Their connection to monadic second order logics was pointed out by Büchi[1] in the 1960s.

Since that time, four key equivalence theorems have been proved. Respectively, finite automata, Büchi automata, finite tree automata, and Rabin automata are expressively equivalent to the weak second order theory of one successor, the second order theory of one successor, the weak second order theory of two successors, and the second order theory of two successors. The theory of one successor has first order part ω, the natural numbers, with the one successor operation $+1$. The theory of two successors has first order part $\{0,1\}^*$, the set of finite strings of 0s and 1s, with operations S_0 and S_1 which respectively tack a 0 and a 1 onto the end of a word. A weak second order theory has, in addition to quantification over first order objects, quantification over finite sets of first order objects. The (full) second order theory has quantification over arbitrary sets of first order objects.

Automata theory is particularly fruitful in terms of equivalence theorems:

[1]Throughout this Dissertation, we will only consider monadic second order theories, so the term monadic will be omitted.

notions of regular expression exist for these classes, various modifications of these automata are known to be expressively equivalent, and we have the Myhill-Nerode Theorem, which provides an abstract characterization of these languages.

CHAPTER 2

GENERATING THE REGULAR FUNCTIONS AND KROHN-RHODES

In this chapter we seek to produce a small generating set for the regular functions, those functions of words which can be expressed in the regular word logic. The main result is that essentially any regular function can be computed by a two-pass process: first scanning the input from left to right, leaving behind some data, and then scanning the input and data left behind from right to left, outputting the value of the function as we go. We then provide an interpretation of the Krohn-Rhodes theorem which allows us to break up our generating set even further. For the convenience of the reader, a simplified version of the proof of the Krohn-Rhodes Theorem from Ginzburg [5] is also presented.

2.1 Moore Machines

The Krohn-Rhodes Theorem concerns itself with *finite state transducers*, an abstraction of systems that:

- Accept inputs from a discrete set at discrete times,
- Retain some memory about previous inputs, which updates whenever an input is read,
- For each input read, produce some output from a discrete set based on the input and memory.

This provides us with an abstraction of synchronous (as opposed to those that update continuously), digital (as opposed to those that deal with analog values) systems.

Originally proved in [7], the Krohn-Rhodes theorem itself allows us to decompose arbitrary finite state transducers into a cascade of transducers from a small generating set. Computational implementations of this decomposition are available [3]. The Krohn-Rhodes Theorem can be used to analyze the rough behavior of automata, providing applications to Artificial Intelligence [4].

We formalize finite state transducers as follows:

Definition. *A* Moore Machine *is a tuple:* $(\Sigma, Q, q_0, \Gamma, \delta, \epsilon)$.

- Σ *is a finite set of input characters (alphabet).*
- Q *is a finite set of states.*
- $q_0 \in Q$ *is the initial state.*
- Γ *is a finite set of output characters.*
- $\delta : Q \times \Sigma \to Q$ *is the transition function.*
- $\epsilon : \Sigma \to \Gamma$ *is the output function.*

For convenience, we denote the map $x \mapsto \delta(a, x)$ *by* δ_a.

Given an input word $w \in \Sigma^*$, *we construct a run* r *and output* o *such that:*

- $r[0] = q_0$.
- $r[i+1] = \delta(r[i], w[i])$ *for* $0 \leq i < |w|$.
- $o[i] = \epsilon(r[i])$, *for* $0 \leq i \leq |w|$.

Where our indexing notation is such that:

$$w = w[0], \ldots, w[|w| - 1].$$

The correct way of thinking about Moore Machines is that each input acts as a transition between one state and the next, or that each state is the state between inputs. A proper representation would have input characters half a step offset from states. Outputs are simply a product of the state the automaton is in and so should be in step with the states. However, representing the sequences of input characters and states as words requires a choice of direction to shift half a step. A considerable effort has been made to pick the option to result in the cleanest presentation. In this paper, the input character $w[i]$ tells the device how to transition from state $r[i]$ to state $r[i+1]$ [1].

Example 1. *Here we show below how inputs, states, and outputs, respectively, line up according to our notation.*

$$\begin{tikzcd}
a \arrow[d, dash] \arrow[dr] & b \arrow[d, dash] \arrow[dr] & b \arrow[d, dash] \arrow[dr] & b \arrow[d, dash] \arrow[dr] & a \arrow[d, dash] \arrow[dr] & \\
q_0 \arrow[d] & q_1 \arrow[d] & q_2 \arrow[d] & q_1 \arrow[d] & q_2 \arrow[d] & q_3 \arrow[d] \\
o_0 & o_1 & o_0 & o_1 & o_0 & o_2
\end{tikzcd}$$

Figure 2.1: Alignment Convention for Moore Machines

We will typically consider Moore Machines that simply output their states:

Definition. *A Moore Machine is said to be* transparent *if* $\Gamma = Q$ *and* ϵ *is the identity. In this case, we present our Moore Machine as a tuple:* (Σ, Q, q_0, δ).

We can interpret Moore Machines as functions from input words to output words:

Definition. *Given a Moore Machine* $M = (\Sigma, Q, q_0, \Gamma, \delta, \epsilon)$*, and word* $w \in \Sigma^*$*, we denote by* $M(w)$ *the output of* M *on input* w*, in* Γ^*.

[1]This is as opposed to the input character $w[i]$ telling the device how to transition from state $r[i-1]$ to state $r[i]$. This alternative is not uncommon.

Moore Machines, however, represent only a small subset of those functions on words which we can reason about using finite automata.

Definition. *Given two alphabets, Σ, Γ, a function on words $f : \Sigma^* \to \Gamma^*$ is said to be:*

- length-preserving *if for any word w, $|f(w)| = |w|$.*
- causal *if $f(w)[i]$ depends only on $w[0], \ldots, w[i]$.*
- strictly causal *if $f(w)[i]$ depends only on $w[0], \ldots, w[i-1]$.*
- character-wise *if $f(w)[i]$ depends only on $w[i]$.*

Definition. *Define the following useful functions for dealing with words:*

- *Let S_a denote the* successor-a *function which appends the character a to the end of a word.*
- *Let Trunc denote the function which removes the last character of a word.*
- *Let Rest denote the function which removes the first character of a word.*

Proposition 1. *Given a Moore Machine M, the function M computes is strictly causal, and increases length by 1.*

We may find it more convenient to deal with length preserving versions of this function:

Definition. *Given a Moore Machine $M = (\Sigma, Q, q_0, \Gamma, \delta, \epsilon)$, and word $w \in \Sigma^*$, we denote by:*

- *$M^{\mathrm{Trunc}}(w)$ the output of M on input w with the last state removed.*
- *$M^{\mathrm{Rest}}(w)$ the output of M on input w with the first state (the start state) removed.*

Proposition 2. *Given a Moore Machine M, the function M^{Trunc} is strictly causal and length preserving, and the function M^{Rest} is causal and length preserving.*

In the next section, we will explore other kinds of functions of which can be reasoned about using finite automata.

2.2 Regular Functions

The notion of finite automaton, or finite state recognizer, is more commonly studied than the finite state transducer. *Finite state automata* are an abstraction of systems that:

- Accept inputs from a discrete set at discrete times,
- Retain some memory about previous inputs, which updates whenever an input is read,
- Having finished reading a sequence of inputs, either *accepts* or *rejects*.

Just as finite state transducers can be interpreted as functions on words, finite state automata can be interpreted as predicates on words, returning a boolean value after having read in a word. Their predicative nature means that finite state automata are more convenient to use in applications to formal logic. On the other hand, real world systems are more often interested in transforming inputs, and so are better represented by finite state transducers.

Finite automata give us a notion of a regular set or regular event, a collection of words or sequences of inputs which are exactly those which some finite automaton accepts. Once we define what it means to represent a function of words with an automaton, this will give us a notion of regular function.

We formalize finite state recognizers as follows:

Definition. *A* finite state automaton *is a tuple:* $(\Sigma, Q, I, \delta, F)$.

- Σ *is a finite set of input characters (alphabet).*
- Q *is a finite set of states.*
- $I \subseteq Q$ *is the set of initial states.*
- $\delta : Q \times \Sigma \to Q$ *is the transition relation.*
- $F \subseteq Q$ *is the set of final states.*

Given an input word $w \in \Sigma^*$*, we say that* $r \in Q^*$ *is a run on input* w *if:*

- $r[0] \in I$.
- $\delta(r[i], w[i], r[i+1])$ *for* $0 \leq i < |w|$.

We say that w *is* accepted *if there is a run* r *on input* w *such that* $r[|w|] \in F$.

Definition. *A finite state automaton* $A = (\Sigma, Q, I, \delta, F)$ *is* deterministic *if:*

- I *is a singleton.*
- δ *is a function from* $Q \times \Sigma \to Q$*, that is, given a* $q \in Q$*, and* $a \in \Sigma$*, there is a unique* $q' \in Q$ *such that* $\delta(q, a, q')$.

By default, we say that A *is nondeterministic.*

A well known theorem of finite automata is that:

Proposition 3. *If R is the set of accepted inputs of some automaton, then it is also the set of accepted inputs of some deterministic automaton.*

In either case, we say that R is *regular*.

A common convention in logic is to identify a function f with the relation R_f which consists of all pairs of the form $(x, f(x))$, or, for n-ary functions,

$$(x_0, \ldots, x_{n-1}, f(x_0, \ldots, x_{n-1})),$$

for x in the domain of f. As such, we will be able to define a regular function as a relation which is regular and also a function. The question now is how to input multiple words, especially multiple words of different lengths, to a finite automaton.

We introduce the $\mathrm{Tuplefy}$ map to merge words together in parallel so they can be read by an automaton. For words of different lengths, we add a dummy character $\#$.

Definition. *Define the map*

$$\mathrm{Tuplefy} : \Sigma_0^* \times \cdots \times \Sigma_{n-1}^* \to ((\Sigma_0 \cup \{\#\}) \times \cdots \times (\Sigma_{n-1} \cup \{\#\}))^*,$$

Which satisfies:

$$\mathrm{Tuplefy}(w_0, \ldots, w_{n-1})[i]_j = \begin{cases} w_j[i] & \textit{it exists} \\ \# & \textit{otherwise} \end{cases}$$

and $|\mathrm{Tuplefy}(w_0, \ldots, w_{n-1})| = \max_i |w_i|$.

Example 2. *Here we show below how* $\mathrm{Tuplefy}$ *combines words together to one word:*

w_0	a	b	b	a	
w_1	a	b			
w_2					
w_3	b	b	b		
w_4	a	a	a	a	a
$\mathrm{Tuplefy}(w_0, w_1, w_2, w_3, w_4)$	$\begin{pmatrix} a \\ a \\ \# \\ b \\ a \end{pmatrix}$	$\begin{pmatrix} b \\ b \\ \# \\ b \\ a \end{pmatrix}$	$\begin{pmatrix} b \\ \# \\ \# \\ b \\ a \end{pmatrix}$	$\begin{pmatrix} a \\ \# \\ \# \\ \# \\ a \end{pmatrix}$	$\begin{pmatrix} \# \\ \# \\ \# \\ \# \\ a \end{pmatrix}$

Figure 2.2: Behavior of Tuplefy on Sample Input

Now we can define the notion of regular relation and regular function:

Definition. *An n-ary relation $R \subseteq \Sigma_0^* \times \cdots \times \Sigma_{n-1}^*$ is* regular *if there is a finite automaton A with input alphabet $(\Sigma_0 \cup \{\#\}) \times \cdots \times (\Sigma_{n-1} \cup \{\#\})$ such that:*

$$R(w_0, \ldots, w_{n-1}) \iff A \textit{ accepts } \mathrm{Tuplefy}(w_0, \ldots, w_{n-1}).$$

An n-ary function $f : \Sigma_0^* \times \cdots \times \Sigma_{n-1}^* \to \Sigma_n^*$ is regular *if there is a finite automaton A with input alphabet $(\Sigma_0 \cup \{\#\}) \times \cdots \times (\Sigma_n \cup \{\#\})$ such that:*

$$f(w_0, \ldots, w_{n-1}) = w_n \iff A \textit{ accepts } \mathrm{Tuplefy}(w_0, \ldots, w_n).$$

Regular relations and functions are a key part of the analysis of various automaton logics. For instance:

Definition. *Given a finite alphabet Σ, let $\mathcal{W}_\Sigma = (\Sigma^*, \leq, =_{el}, S_a|_{a \in \Sigma})$ where:*

- *$\leq$ is the prefix relation on words,*
- *$=_{el}$ is the equal length relation on words,*

- *S_a is the* Successor-a *unary operation, which appends an a onto the end of a word.*

We call $\mathcal{W}_\Sigma$ the regular word logic over Σ.

Proposition 4. *If R is a relation on Σ^*, the following are equivalent:*

- *R is regular,*
- *R is given by a formula ϕ in the language of $\mathcal{W}_\Sigma$.*

Hence our interest in regular functions. If $\phi(x_0, \ldots, x_n)$ is a formula in the language of $\mathcal{W}_\Sigma$ such that:

$$\forall x_0, \ldots, x_{n-1} \exists x_n : \phi(x_0, \ldots, x_n),$$

Then, since lexicographic ordering $<_L$ is regular and well-founded, we can restrict to the regular relation:

$$\psi(x_0, \ldots, x_n) \iff \phi(x_0, \ldots, x_n) \wedge \nexists y : [y <_L x_n \wedge \phi(x_0, \ldots, y)],$$

Which is also a function. Such a restriction is called a *Skolem function*.

Finally, in this section, we connect Moore Machines and Finite Automata:

Proposition 5. *Given a Moore Machine $M = (\Sigma, Q, q_0, \Gamma, \delta, \epsilon)$, the functions M, M^{Trunc}, and M^{Rest} are regular.*

Proof. We can construct a deterministic finite automaton that keeps track of two pieces of information: the state we expect the Moore Machine to be in at any particular point, and a boolean value to keep track of whether the proposed output has so far been correct, additionally, in the case of M, we should expect a final element of the form $\begin{pmatrix} \# \\ o \end{pmatrix}$ and a small amount of information must be kept track of to handle this correctly.

For example, for M^{Trunc}, let:

$$A = (\Sigma \times \Gamma, Q \times \{0,1\}, (q_0, 0), \delta', Q \times \{0\}),$$

Where:

$$\delta'((q,i),(a,o)) = \left(\delta(q,a), \begin{cases} 0 & i = 0 \wedge \epsilon(q) = o \\ 1 & \text{otherwise} \end{cases} \right).$$

$\square$

Proposition 6. *Every strictly causal, length-preserving, regular function is given by M^{Trunc} for some Moore Machine M. Every causal, length-preserving function is given by M^{Rest} for some Moore Machine M.*

Proof. We prove this for a binary, strictly causal, length-preserving, regular function f. The proof is nearly identical in the general case. Let $f : \Sigma \to \Gamma$ be given. By our equivalence theorem above, there is a formula $\phi(w_0, w_1)$ in the language $\mathcal{W}_{\Sigma \cup \Gamma}$ such that

$$\phi(w_0, w_1) \iff f(w_0) = w_1.$$

Since f is a strictly causal function, the first $n-1$ characters of the input determine the nth character of the output. By simple tricks in $\mathcal{W}_{\Sigma\cup\Gamma}$, we can construct a formula ϕ_u for each $u \in \Gamma$ such that ϕ_u is true of exactly those sequences of characters that produce an output of u in the next place. These ϕ_u describe a collection of regular sets which partition all of Σ^*. Let A_u be a finite automaton that recognizes the corresponding collection.

Now we construct our Moore Machine M. It should run each of the A_u in parallel to determine its state. Since the A_u recognize disjoint collections, exactly one of the A_u will be in an accept state at any time. The ϵ function for our Moore Machine will take in the tuple of states for the A_u and output the one which was in an accept state. It suffices now to check that M^{Trunc} is identically f.

The proof for M^{Rest} is similar. □

Of course, there are plenty of other regular functions. Take for instance, the Rest function, which removes the first character of a string, shifting to the left. Alternately, a function which outputs 00 or 01 depending on whether there are an even or odd number of 0s in the input. Moore Machines allow us to construct functions which transmit information only to the right (towards the end of the inputs), but we will need a new method to transmit information to the left as in these examples.

2.3 Determinization and Harvesting

In this section, we introduce a notion of a Reverse Moore Machine, and provide a technique for decomposing a length-preserving regular function as a multi-variable composition of a Moore Machine function, a Reverse Moore Machine function, and character-wise maps to connect them. This will require several stages. First, we will briefly discuss some notation for discussing character-wise functions. Second, we will introduce the notion of a Reverse Moore Machine similar to the Moore Machines introduced in section 1. Third, we will present the decomposition. This decomposition is based on the classical powerset determinization construction, viewed from a novel perspective. In the next section, we will discuss handling general regular functions.

First, some notation for functions which operate character-wise:

Definition. *Given two finite alphabets Σ, Γ, and a function $f : \Sigma \to \Gamma$, we call the function $\mathrm{Cw}_f : \Sigma^* \to \Gamma^*$ which applies f to each character of the input, the* character-wise f map.

If f is an n-ary function for $n > 1$, we can also make sense of Cw_f. Let

$$f : \Sigma_0 \times \cdots \times \Sigma_{n-1} \to \Gamma.$$

Then we define the partial function

$$\mathrm{Cw}_f : \Sigma_0^* \times \cdots \times \Sigma_{n-1}^* \to \Gamma^*,$$

which takes in inputs of all the same lengths and produces an output of the same length, where:

$$\mathrm{Cw}_f(w_0, \ldots, w_{n-1}) = \mathrm{Cw}_f(\mathrm{Tuplefy}(w_0, \ldots, w_{n-1})).$$

Noting that Tuplefy *does not produce characters with # in them for equal-length inputs.*

Of course, Cw_f is causal and length preserving (and reverse-causal, when we define the notion). Every Moore Machine function can be written as the corresponding function for the corresponding transparent Moore Machine composed with a bitwise application of its ϵ function.

We now define some notation for dealing with Reverse Moore Machines, analogous to our notation established previously.

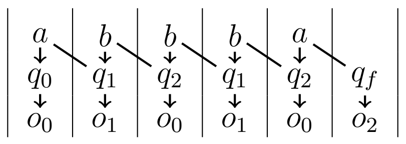
Definition. *A* Reverse Moore Machine *is a tuple:* $(\Sigma, Q, q_f, \Gamma, \delta, \epsilon)$.

- *Σ is a finite set of input characters (alphabet).*
- *Q is a finite set of states.*
- *$q_f \in Q$ is the final state.*
- *Γ is a finite set of output characters.*
- *$\delta : Q \times \Sigma \to Q$ is the reverse transition function.*
- *$\epsilon : \Sigma \to \Gamma$ is the output function.*

Given an input word $w \in \Sigma^*$, we construct a run r and output o such that:

- *$r[|w|] = q_f$.*
- *$r[i] = \delta(r[i+1], w[i])$ for $0 \leq i < |w|$.*
- *$o[i] = \epsilon(r[i])$, for $0 \leq i \leq |w|$.*

Example 3. *Here we show below how inputs, states, and outputs, respectively, line up according to our notation.*



a b b b a
q_0 q_1 q_2 q_1 q_2 q_f
o_0 o_1 o_0 o_1 o_0 o_2

Figure 2.3: Alignment Convention for Reverse Moore Machines

To prevent type mismatches, we will denote a Reverse Moore Machine with letters R, P as opposed to letters M, N for Moore Machines.

Definition. *A Reverse Moore Machine is said to be* transparent *if $\Gamma = Q$ and ϵ is the identity. In this case, we present our Reverse Moore Machine as a tuple: (Σ, Q, q_f, δ).*

As before, we can interpret Reverse Moore Machines as functions from input words to output words:

Definition. *Given a Reverse Moore Machine $R = (\Sigma, Q, q_f, \Gamma, \delta, \epsilon)$, and word $w \in \Sigma^*$, we denote by:*

- *$R(w)$ the output of R on input w.*
- *$R^{\text{Trunc}}(w)$ the output of R on input w with the last state removed.*
- *$R^{\text{Rest}}(w)$ the output of R on input w with the first state (the start state) removed.*

Analogous to our notions of causal and strictly causal, we have notions of reverse causal and strictly reverse causal:

Definition. *Given two alphabets, Σ, Γ, a function on words $f : \Sigma^* \to \Gamma^*$ is said to be:*

- reverse causal *if $f(w)[i]$ depends only on $w[i], w[i+1], \ldots$.*
- strictly reverse causal *if $f(w)[i]$ depends only on $w[i+1], w[i+2], \ldots$.*

Of course, a function is character-wise iff it is causal and reverse causal.

We also have a notion of reverse deterministic automaton:

Definition. *A finite automaton $A = (\Sigma, Q, I, \delta, F)$ is* reverse deterministic *if:*

- *F is a singleton.*
- *For each q, a, there is a unique q' such that $(q', a, q) \in \delta$.*

A reverse deterministic automaton can also be viewed as a transparent Reverse Moore Machine.

We now have the notation to state our Theorem:

Theorem 1. *Any length-preserving regular function can be written as a multivariable composition of the form:*

$$f(w) = R^{\mathrm{Trunc}}(\mathrm{Cw}_{\mathrm{Pair}}(w, M^{\mathrm{Trunc}}(w))).$$

It's worth noting that $\mathrm{Cw}_{\mathrm{Pair}}$ has the same action here as $\mathrm{Tuplefy}$. We've chosen to use $\mathrm{Cw}_{\mathrm{Pair}}$ here to indicate that we're not using the length-padding features of $\mathrm{Tuplefy}$.

We need to prove a few lemmas first.

Lemma 1. *Given a length-preserving regular function $f : \Sigma^* \to \Gamma$, there is an automaton B with state set Q and map $\epsilon : Q \to \Gamma$ such that $f(w)$ is $\mathrm{Trunc} \circ \mathrm{Cw}_\epsilon$ applied to any run of B on input w.*

Proof. Given our length-preserving function $f : \Sigma^* \to \Gamma^*$, let:

$$C = ((\Sigma \cup \{\#\}) \times (\Gamma \cup \{\#\}), Q, I, \delta, F)$$

Witness the regularity of f. Since C automatically rejects any inputs with a $\#$ in them, we can restrict it to:

$$A = (\Sigma \times \Gamma, Q, I, \delta, F)$$

Recognizing the same set of inputs.

Construct an automaton:

$$B = (\Sigma, Q \times \Gamma, I \times \Gamma, \delta', F \times \Gamma),$$

Where:

$$((q, o), a, (q', o')) \in \delta' \iff (q, (a, o), q') \in \delta$$

B is nondeterministic. I assert that runs of B on input w will necessarily have Γ component $S_o(f(w))$, for some $o \in \Gamma$. Firstly, it should be clear that if r is an accepting run of A on input $\mathrm{Tuplefy}(w, f(w))$, then $\mathrm{Tuplefy}(r, S_o(f(w)))$ is an accepting run of B on input w for any $o \in \Gamma$.

Now, suppose $\mathrm{Tuplefy}(r', g)$ is an accepting run of B on input w. Then it is easy to check that r' is an accepting run of A on input $\mathrm{Tuplefy}(w, \mathrm{Trunc}(g))$. Since A witnessed f being a regular function, $\mathrm{Trunc}(g)$ must be $f(w)$. This completes the proof for ϵ taking the Γ component of the states of B. $\square$

Definition. *Given a nondeterministic automaton $A = (\Sigma, Q, I, \delta, F)$, define its* determinization*:*

$$\det(A) = (\Sigma, \mathcal{P}(Q), \{I\}, \delta', \{E \subset Q : E \cap F \neq \emptyset\}),$$

Where:

$$\delta'(K, a) = \{q' \in Q | \exists q \in K : \delta(q, a, q')\}.$$

Typically throughout this chapter we will be concerned with the determinization as a transparent Moore Machine, so the set of final states doesn't matter much.

Definition. *Given a nondeterministic automaton $A = (\Sigma, Q, I\delta, F)$, define its* harvester*:*

$$\mathrm{harv}(A) = (\Sigma \times \mathcal{P}(Q), Q \cup \{q_F\}, Q \cup \{q_F\}, \delta', \{q_F\}),$$

Where:

- *The q such that $\delta'(q, (a, K), q')$ (for $q' \in Q$) is given by the least $q'' \in K$ such that $\delta(q'', a, q')$ or q_F if none exist.*

- *The q such that $\delta'(q,(a,K),q_F)$ is given by finding the least $q' \in F$ such that there is a $q'' \in K$ such that $\delta(q'',a,q')$ and then having q be the least $q'' \in K$ such that $\delta(q'',a,q')$. If no such $q' \in F$ exists, then q is given by q_F.*

The powerset determinization of an automaton A produces an automaton that keeps track of, at every position, the set of states of A which are reachable through some sequence of transitions, having read the input up to that point. However, not every one of these reachable states necessarily shows up in some accepting run: it may be that being in one state now means later on having to be in another state which we cannot transition out of, or that being in a state now dooms us to being in a reject state once we have finished reading the input. In order to use our determinization to find an accepting run of the original automaton, provided there is an accepting run, we need to start at the end and work our way backwards, all the while staying within states that we know can be traced through a sequence of transitions back to a start state at the beginning.

Specifically, if we know we're in a state q which is reachable through some sequence of transitions after having read $S_a(w)$, then there must be at least one state q' which is reachable through some sequence of transitions after having read w such that reading a takes us from state q' to state q.

It is necessary to introduce an additional dummy state q_F to start out in to make sure our harvester automaton is reverse-deterministic. Although our application of the harvester automaton will see only pairs of the form (a,K) for a some symbol being read by our original automaton and K the set of states reachable immediately prior to reading that specific a, our automaton should be prepared to read in arbitrary input pairs. An invalid input will cause the reverse deterministic automaton to go into the dummy q_F state. Additionally, a cheap fix is necessary to account for the fact that we don't know what state the run we're trying to produce ends on and we need a single "final" state for our reverse deterministic automaton to "start out" in. The dummy q_F state represents all final states which are reachable. Truncating the run will remove this dummy state.

Lemma 2. *Suppose we have a nondeterministic automaton A, and valid input w. Let p be the run of $\det(A)$ on input w. I claim that $\mathrm{harv}(A)$ on input $\mathrm{Tuplefy}(w, \mathrm{Trunc}(p))$ will have run r, an accepting run of A on input w.*

Proof. It suffices to show that the only occurrence of q_F in r is as the final character. By construction, $\mathrm{harv}(A)$ will satisfy the transition relations. As mentioned before the construction of $\mathrm{harv}(A)$ also prevents backwards transitioning into the q_F state for this particular input, since a reachable state can always be traced back to a reachable state. $\square$

Combining our two lemmas, given a regular, length-preserving function f, we have a nondeterministic automaton B which takes in a word w and has a run r which projects to $f(w)$. By our second lemma:

$$r = \mathrm{harv}(B)^{\mathrm{Trunc}}(\mathrm{Cw}_{\mathrm{Pair}}(w, \det(B)^{\mathrm{Trunc}}))$$

By modifying the outer Reverse Moore Machine, we can throw in the appropriate projection to its output map to produce $f(w)$. This completes the proof.

This is a remarkable result. Every length-preserving regular function can be computed in a two-step process: one pass forwards through the input leaving behind some information as we do, then a pass backwards through the input with this additional information to directly produce the output. Two passes suffice; having more passes doesn't increase our expressive power.

2.4 Length Modification

We're very close to generating all the regular functions, but we're not quite there. The only functions we have dealt with so far were length-preserving. In this section, we see that most of the interesting behavior was already captured in the length-preserving case.

Proposition 7. *Suppose $f : \Sigma \to \Gamma$ is a regular function. Then there is a fixed constant c associated to f such that $f(w)$ is no longer than $c + |w|$ for every w.*

The proof is based on the pumping lemma.

Proof. Let A be a deterministic automaton with c states accepting exactly words of the form $\mathrm{Tuplefy}(w, f(w))$. Choose a specific w and suppose $f(w)$ is longer than $c + |w|$. Imagine what happens as A reads in $\mathrm{Tuplefy}(w, f(w))$, specifically, after w is finished, and A is reading in characters of the form $\begin{pmatrix} \# \\ o \end{pmatrix}$ for some $o \in \Gamma$. Because there are more positions like this than there are states of A, by the pigeonhole principle some two positions will have the same state, say at positions i and j. Note however that if we remove all positions in $\mathrm{Tuplefy}(w, f(w))$ between i and j (including i, excluding j) we still have an accepting run, of the form $\mathrm{Tuplefy}(w, g)$ for some g strictly shorter than $f(w)$. This contradicts our assumption that A accepted exactly words of the form $\mathrm{Tuplefy}(w, f(w))$. $\square$

From this, we can also show that for any n-ary regular function f, there is a fixed constant c associated to f such that $f(w_0, \ldots, w_{n-1})$ is no longer than c plus the length of the maximum input.

Note that the automaton A in our proof never encounters the input character $\begin{pmatrix} \# \\ \# \end{pmatrix}$. As such, we may assume that this character acts as the identity on the states of A and still have an automaton witnessing the regularity of f. This automaton recognizes all pairs of the form $\mathrm{Tuplefy}(S_{\#}^{c}(w), f(w))$. ($S_{\#}^{c}$ simply represents a c-fold composition of the $S_{\#}$ function).

As such, the function which takes $S_{\#}^{c}(w)$ to $S_{\#}^{d}(f(w))$ for $d = |f(w)| - |w|$ is a length-preserving regular function g, and thus can be written as a composition of character-wise maps, truncated Moore Machines, and truncated Reverse Moore Machines as in Theorem 1. Since f can be written as:

$$f(w) = \mathrm{Unpad}(g(S_{\#}^{c}(w))),$$

Where Unpad removes final # characters. Note that we do not know how many final # characters there will be. For functions which reduce length, the number will be more than c and could be as much as $c + |w|$. One might be tempted to try to replace Unpad with some function like $S_{\#}^{-1}$ (or, even less suited to the task, Trunc), which either removes a single final # or leaves the word alone if it cannot. However, since there are regular functions which take words of arbitrary length and reduce them to length 1, we need a generator that can produce unbounded shortening as well.

Note that if f is n-ary for $n > 1$, we can use what we've already shown to see that:

$$f(w_0, \ldots, w_{n-1}) = \mathrm{Unpad}(g(S_{\#^n}^{c}(w))),$$

For some regular, length-preserving function g.

As such, we have the following theorem:

Theorem 2. *Any regular function can be written as a multivariable composition of:*

- *Truncated Moore Machines,*
- *Truncated Reverse Moore Machines,*
- *Character-wise maps,*
- Tuplefy *(allowing us to generate multiary character-wise maps),*
- S_a *for various* a*,*
- Unpad.

It's worth noting here that our generating set is infinite. Specifically, there are an infinite number of Moore Machines and Reverse Moore Machines. There are also an infinite number of Character-wise maps, but this isn't essential – one could use encoding methods to work purely with a single two-character alphabet (plus, optionally, the dummy character $\#$).

The infinitude of our generating set, however, is essential. The easiest way to see this is to talk about period introduction. Provided the input to a regular function has a sufficiently long periodic portion in the middle, the output of the regular function will also have a long periodic portion in the middle (it suffices to verify this of the generators above). What's more, the period of the periodic portion of the output can only have prime factors which show up either in the periods of the periodic portions of the inputs or which are smaller than the number of states of the associated automaton to the regular function. However, one can easily build regular functions which introduce any prime factor into the periodicity of their inputs, so we must not allow any bound on the sizes of associated automata to regular functions in our generating set. We summarize this result as follows:

Proposition 8. *Any set of regular functions which generates all regular functions under multivariable composition must be infinite.*

We conclude this chapter with a discussion of the Krohn-Rhodes Theorem. First, we will provide a proof of the Krohn-Rhodes Theorem adapted from Ginzburg [5]. Then we will use the Krohn-Rhodes Theorem to decompose our Moore Machine and Reverse Moore Machine generators into smaller, simpler generators. A final short discussion will clean up our set of generators, followed by proposed future research.

2.5 The Krohn-Rhodes Theorem

In this section, we present the Krohn-Rhodes Theorem as adapted to the context of multivariate composition of regular functions. The original proof of the Krohn-Rhodes Theorem, in [7], was presented in terms of wreath products of semigroups. More modern presentations of the Krohn-Rhodes Theorem typically present it in terms of the cascade product of finite state transducers.

The cascade product of two transducers M_1, M_2 is a system consisting of both machines. First, machine M_2 reads in both the input to the system and the current state of M_1 to update its state. When it has finished, machine M_1 updates its state based only on the input to the system. Finally, an output is produced based on the states of M_1 and M_2. This reflects the reality of systems where updating the states of our machines takes a small but appreciable amount of time. In a well designed system, M_2 should not have to wait for M_1 to finish its update before it can update its state. As such, M_2 uses the state of M_1 prior to reading the input to update.

The Krohn-Rhodes Theorem separates out two extremes of behavior for finite automata. In general, reading in an input character induces a function on the states of the automaton. This function may map two states to the same state or to separate states. At one extreme, it may act as a *permutation* in which case it maps all states to separate states. In this case, it is possible to undo this action. We can recover the state before reading a character which acts as a permutation, provided we know which character the automaton read. At the other extreme, an input character may act as a *reset* in which case it maps all states to the same state. In this case all information about the previous state is lost.

Definition. *A Moore Machine or deterministic automaton is said to be:*

- *A* permutation automaton *if each of its inputs acts as a permutation on its states,*
- *A* reset automaton *if each of its inputs acts as a reset or the identity on its states,*
- *A* permutation-reset automaton *if each of its inputs acts as a permutation or a reset on its states.*

We now state the Krohn-Rhodes theorem, in a bit of an unusual fashion:

Theorem 3 (Krohn-Rhodes)**.** *Given a transparent Moore Machine M, we can write its truncated action M^{Trunc} as a multivariable composition of truncated actions of permutation-reset Moore Machines $M_0, \ldots, M_{n-1}$ for n the number of states of M, and a final character-wise map f. What's more, this composition takes on a fairly simple* *form. Let:*

$$\begin{aligned} w_0 =& M_0^{\text{Trunc}}(w), \\ w_1 =& M_1^{\text{Trunc}}(\text{Tuplefy}(w, w_0)), \\ w_2 =& M_2^{\text{Trunc}}(\text{Tuplefy}(w, w_0, w_1)), \\ &\vdots \\ w_{n-1} =& M_{n-1}^{\text{Trunc}}(\text{Tuplefy}(w, w_0, \ldots, w_{n-2})), \end{aligned}$$

Then:

$$M^{\text{Trunc}}(w) = \text{Cw}_f(w_0, \ldots, w_{n-1}).$$

Every map in this composition is length-preserving. Of course we can write this as simply one large multivariable composition of a character-wise map and truncated actions of permutation-reset transparent Moore Machines, but this is unwieldy to write down. The above also presents an efficient way of computing the composition, although readers concerned with efficiency are encouraged to look into the Holonomy decomposition [3].

Our proof is inductive: we show that for every transparent Moore Machine M, there's another transparent Moore Machine M' that keeps track of a state M is not in in a permutation-reset way. This reduces the amount of information we need to keep track of by one state, and we can keep doing this until we've kept track of all the information to know what state M is in.

The proof in [5] allows for the possibility that we can keep track of several states our automaton M is not in in a permutation-reset way at the same time, as opposed to one at a time in the proof below. This is more efficient, but adds needless complexity to the proof.

The key piece of our construction is the Permutation-Reset Lemma, below.

Lemma 3 (Permutation-Reset Lemma)**.** *Given two finite ordered sets of the same size I, J, and map between them f, there is a map $g : I \to J$ such that:*

- *g either acts as:*
  - *A bijection from I to J (a permutation on the position indices),*
  - *Or has singleton image (a reset on position indices),*
- *And for $x, y \in I$, with $x \neq y$, we have $f(x) \neq g(y)$.*
- *For any $x \in I$, we have f maps elements of $I \setminus \{x\}$ to $J \setminus \{g(x)\}$.*

Proof. Suppose f does act as a bijection. Then $g = f$ is a permutation and satisfies the inequality condition.

Suppose f does not act as a bijection. Then g which maps everything to the smallest element of J which is not in the image of f has singleton image and satisfies the inequality condition.

The third condition is just a rephrasing of the second, but will come in handy later on. □

A specific application of the Permutation-Reset Lemma is that we can have a transparent Moore Machine that keeps track of a state our original transparent Moore Machine is not in:

Lemma 4. *Given a transparent Moore Machine $M = (\Sigma, Q, q_0, \delta)$, there is a permutation-reset transparent Moore Machine $\overline{M}$ with the same state set such that on input w, the state of M at any one time is not the state of $\overline{M}$.*

Proof. Assign a natural ordering to Q. Define

$$\overline{M} = (\Sigma, Q, \overline{q}_0, \overline{\delta}),$$

Where $\overline{q}_0$ is the smallest element in Q which is not q_0, and $\overline{\delta}(q, a)$ is given by:

- $\delta(q, a)$ if a acts as a permutation.
- Otherwise, the smallest $q' \in Q$ which is not in the image of any state under the action of a.

If the action of a was a permutation originally, it is still a permutation in our new automaton. This permutation not only takes us from the state our automaton is in before reading a to the state afterwards, but also from a state our automaton is not in before reading a to a state our automaton is not in afterwards. In the second case, note that the choice of q' does not depend on q, so this action is a reset. Obviously, it takes us from a state our original automaton is not in before reading a to a state our automaton is not in after reading a. □

Lemma 5. *Given transparent Moore Machines $M, \overline{M}$ as above with state sets Q, there is a third transparent Moore Machine:*

$$\widehat{M} = (\Sigma \times Q, \{0, \ldots, |Q| - 2\}, \hat{\imath}_0, \widehat{\delta}),$$

Such that for any input w, if M on reading w winds up in state q, and $\overline{M}$ on reading w winds up in state $\overline{q}$ then $\widehat{M}$ on reading $\text{Tuplefy}(w, \overline{M}^{\text{Trunc}}(w))$ *will wind up in state $\hat{\imath}$, where q is the element in position*[2] *$\hat{\imath}$ of $Q \setminus \{\overline{q}\}$.*

[2]To maintain notational consistency within this paper, where ordered collections are indexed starting with 0, we refer to the first element of a set as being in position 0, and generally the $i+1$st element of a set as being in position i. To avoid confusion, we will endeavor to avoid using the notation "ith element", instead using notation "the element in position i".

Proof. Let $\widehat{M} = (\Sigma \times Q, \{0, \ldots, |Q|-2\}, \hat{\imath}_0, \widehat{\delta})$ where $\hat{\imath}_0$ is the index of q_0 in $Q \setminus \{\overline{q}_0\}$ and $\widehat{\delta}(i, (a, \overline{q}))$ is computed by:

- Computing $q' = \overline{\delta}(\overline{q}, a)$. This is the state $\overline{M}$ says that M is not in after reading a.
- Computing q, the ith element of $Q \setminus \{\overline{q}\}$. This is the state of M prior to reading a.
- Return j, the index of $\delta(q, a)$ in $Q \setminus q'$.

The above construction is designed specifically to satisfy the conclusion. □

As such, we have that

$$M^{\text{Trunc}}(w) = \text{Cw}_p(\overline{M}^{\text{Trunc}}(w), \widehat{M}^{\text{Trunc}}(\text{Tuplefy}(w, \overline{M}^{\text{Trunc}}(w))))$$

Where $p(\overline{q}, i)$ is the ith element of $Q \setminus \{\overline{q}\}$.

Finally, we prove the Krohn-Rhodes Theorem:

Proof. By induction on the number of states of M.

Base Case: If M has one state, then f in the composition is 0-ary, and we can have it just output that constant state.

Inductive Case: Given our M, we can write:

$$M^{\text{Trunc}}(w) = \text{Cw}_p(\overline{M}^{\text{Trunc}}(w), \widehat{M}^{\text{Trunc}}(\text{Tuplefy}(w, \overline{M}^{\text{Trunc}}(w)))),$$

With $\overline{M}$ permutation-reset. By our inductive hypothesis, we can write $\widehat{M}^{\text{Trunc}}(w)$, which has one fewer state than M, as:

$$\widehat{M}^{\text{Trunc}}(w) = \text{Cw}_f(w_1, \ldots, w_{n-1}),$$

Where:

$$\begin{aligned}
w_1 =& M_1^{\text{Trunc}}(w)\\
w_2 =& M_2^{\text{Trunc}}(\text{Tuplefy}(w, w_1)))\\
w_3 =& M_3^{\text{Trunc}}(\text{Tuplefy}(w, w_1, w_2)\\
&\vdots\\
w_{n-1} =& M_{n-1}^{\text{Trunc}}(\text{Tuplefy}(w, w_1, \dots, w_{n-2}))
\end{aligned}$$

As such, $\widehat{M}^{\text{Trunc}}(\text{Tuplefy}(w, \widehat{M}^{\text{Trunc}}(w)))$ is given by just plugging in:

$$\widehat{M}^{\text{Trunc}}(\text{Tuplefy}(w, \overline{M}^{\text{Trunc}}(w))) = \text{Cw}_f(w_1, \dots, w_{n-1}),$$

Where:

$$\begin{aligned}
w_1 =& M_1^{\text{Trunc}}(\text{Tuplefy}(w, \overline{M}^{\text{Trunc}}(w))),\\
w_2 =& M_2^{\text{Trunc}}(\text{Tuplefy}(\text{Tuplefy}(w, \overline{M}^{\text{Trunc}}(w)), w_1)),\\
w_3 =& M_3^{\text{Trunc}}(\text{Tuplefy}(\text{Tuplefy}(w, \overline{M}^{\text{Trunc}}(w)), w_1, w_2)),\\
&\vdots\\
w_{n-1} =& M_{n-1}^{\text{Trunc}}(\text{Tuplefy}(\text{Tuplefy}(w, \overline{M}^{\text{Trunc}}(w)), w_1, \dots, w_{n-2})).
\end{aligned}$$

Alternately, fiddling with some parentheses in the definitions of our automata:

$$\widehat{M}^{\text{Trunc}}(\text{Tuplefy}(w, \overline{M}^{\text{Trunc}}(w))) = \text{Cw}_f(w_1, \dots, w_{n-1},)$$

Where:

$$\begin{aligned}
w_0 =&\overline{M}^{\text{Trunc}}(w),\\
w_1 =&M_1^{\text{Trunc}}(\text{Tuplefy}(w, w_0)),\\
w_2 =&M_2^{\text{Trunc}}(\text{Tuplefy}(w, w_0, w_1)),\\
w_3 =&M_3^{\text{Trunc}}(\text{Tuplefy}(w, w_0, w_1, w_2)),\\
&\vdots\\
w_{n-1} =&M_{n-1}^{\text{Trunc}}(\text{Tuplefy}(w, w_0, \dots, w_{n-2})).
\end{aligned}$$

In which case:

$$M^{\text{Trunc}}(w) = \text{Cw}_p(\underbrace{\overline{M}^{\text{Trunc}}(w)}_{w_0}, \text{Cw}_f(w_1, \dots, w_{n-1})).$$

We can combine p and f to get a single character-wise function on $w_0, \dots, w_{n-1}$, thus completing the induction. $\square$

The proof is still straightforward if we unwind the induction. In our construction, w_0 is keeping track of a state M is not in, but it may as well be keeping track of an index for a state M is not in. w_1 is keeping track of an index of a state M is not in once we've removed the state in position w_0 from Q. w_2 is keeping track of an index of a state M is not in once we've removed the states w_1 and w_2 are keeping track of from Q. And so forth. We formalize this indexed removal process as follows:

Definition. *Given a positive integer n, an* ordinal removal sequence *for n is a (possibly empty) sequence of positive integers $(k_0, \dots, k_i)$ satisfying:*

$$\begin{aligned}
i <&n,\\
0 \leq k_j <&n - j.
\end{aligned}$$

We interpret an ordinal removal sequence for n as a series of commands operating on an ordered set of size n of the form "remove the $i+1$st smallest element remaining." Note that as elements are removed, there are fewer elements remaining, hence the decreasing upper limit on k_j in the second constraint. Consistent with the rest of this paper, we begin our indexing with 0, so a 0 means remove the smallest element.

Definition. *Given an ordered set L, and an ordinal removal sequence for $|L|, \mathbf{k} = (k_0, \ldots, k_{i-1})$, define $\mathrm{Remove}(L, \mathbf{k})$ recursively:*

- $\mathrm{Remove}(L, ()) = L$,
- $\mathrm{Remove}(L, (k_0, \ldots, k_{i-1}))$ *is given removing the element in position k_i (with order inherited from L, and 0 means remove the smallest element) of* $\mathrm{Remove}(L, (k_0, \ldots, k_{i-2}))$.

Let $\mathcal{O}_{k,n}$ denote the set of ordinal removal sequences for n of length k.

Recall that $^\frown$ is used for concatenation, so:

$$(k_0, \ldots, k_{i-1})^\frown j = (k_1, \ldots, k_{i-1}, j).$$

Example 4. *Consider the ordinal removal sequence for 5: $(0,1,2,1)$ acting on the ordered set (A,B,C,D,E):*

Start:	(A,B,C,D,E)
Remove at position 0:	(B,C,D,E)
Remove at position 1:	(B,D,E)
Remove at position 2:	(B,D)
Remove at position 1:	(B)

Table 2.1: Sample Application of an Ordinal Removal Sequence

We now state the lemma formally that allows us to construct the M_j in the Krohn-Rhodes decomposition.

Lemma 6. *Given a transparent Moore Machine $M = (\Sigma, Q, q_0, \delta)$, and $0 \leq j < |Q| - 1$, there is a permutation-reset transparent Moore Machine*

$$M_j = (\Sigma \times \mathcal{O}_{j,|Q|}, \{0, \ldots, |Q| - j\}, k_0, \delta'),$$

Such that if κ is a word of ordinal removal sequences for $|Q|$ of length j, that is, $\kappa \in (\mathcal{O}_{j,|Q|})^*$, satisfying:

- *At any point i, $\kappa[i]$ does not remove $M(w)[i]$ from Q. That is, $M(w)[i] \in \mathrm{Remove}(Q, \kappa[i])$.*
- *$\kappa[i+1]$ is determined by $\kappa[i]$ and $w[i]$, specifically such that:*
- *The map δ_a (from M) maps states in $\mathrm{Remove}(Q, \kappa[i])$ to states in $\mathrm{Remove}(Q, \kappa[i+1])$.*

Then $o = M_j(\mathrm{Tuplefy}(w, \kappa))$ satisfies:

- *At any point i, $\kappa[i]^\frown o[i]$ does not remove $M(w)[i]$ from Q. That is, $M(w)[i]$ is not in position $o[i]$ of $\mathrm{Remove}(Q, \kappa[i])$, and in particular $M(w)[i] \in \mathrm{Remove}(Q, \kappa[i]^\frown o[i])$.*

Proof. The idea here is that each M_j should keep track of a single entry in an ordinal removal sequence that will remove all elements of Q except the state of our original automaton M at any one particular time. These will be the M_j in our multivariate composition, so they will be reading in both the original input (a single character a from w), and a single index from each of $M_0, \ldots, M_{j-1}$, together forming an ordinal removal sequence ($\kappa[i]$) of length j. Each M_j then keeps track of an index, which, when added on to the end of the ordinal removal sequence does not remove the one state we must not remove, the state of M at that point.

For reference, a picture of the situation is drawn below:

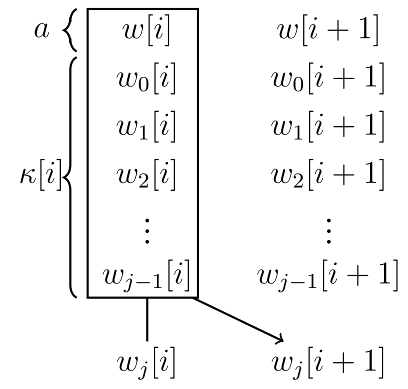


Figure 2.4: Transition Diagram for the Automaton M_j in our Cascade

As we can see in the diagram, M_j will be reading in $w[i]$, the character that takes our automaton M from $M(w)[i]$ to $M(w)[i+1]$, and $\kappa[i]$, the ordinal removal sequence within which we interpret its current state $w_j[i]$. Specifically, $w_j[i]$ will be a position in $\mathrm{Remove}(Q, \kappa[i])$ where we don't find the current state of M, $M(w)[i]$. This will transition M_j into the state $w_j[i+1]$, which must be a position in $\mathrm{Remove}(Q, \kappa[i+1])$ where we don't find the next state of M, $M(w)[i]$.

Note that M_j does not get direct access to $\kappa[i+1]$, but of course we need access to $\kappa[i+1]$ in order to determine the index for $M(w)[i+1]$ in $\mathrm{Remove}(Q, \kappa[i+1])$ so that we can avoid it. Fortunately, if the previous automata, $M_0, \ldots, M_{j-1}$ work in canonical fashions, knowing a and $\kappa[i]$ is enough to determine $\kappa[i+1]$.

To start with, we need to pick the starting state for M_j, k_0, such that k_0 is not an index for the start state of M in $\mathrm{Remove}(Q, \kappa[0])$. Let it be the smallest such index.

By hypothesis, the transition map induced by the character a on the automaton M, δ_a, maps states in $\mathrm{Remove}(Q, \kappa[i])$ to states in $\mathrm{Remove}(Q, \kappa[i+1])$. By the Permutation-Reset Lemma, we can define the transition map for M_j, δ', with $\delta'_{(w[i], \kappa[i])}$ a permutation-reset map for any particular $w[i]$ and $\kappa[i]$ that does the avoiding we require of it. $\square$

Finally, it's worth noting that $\kappa[i]^\frown w_j[i]$, $\kappa[i+1]^\frown w_j[i+1]$ satisfy the requirements on $\kappa[i]$ and $\kappa[i+1]$ in the hypothesis of our lemma. Specifically:

- Our new $w[0]$ and $w_j[i+1]$ were chosen to avoid removing $M(w)[0]$ and $M(w)[i+1]$ from Q.
- $w_j[i+1]$ is determined by $w_j[i], \kappa[i]$, and $w[i]$.
- The map δ_a maps states in $\mathrm{Remove}(Q, \kappa[i]^\frown w_j[i])$ to states in $\mathrm{Remove}(Q, \kappa[i+1]^\frown w_j[i+1])$. This is immediate, looking at the third condition on the function the Permutation-Reset Lemma constructs.

As such the M_j in our proof are the same as the M_j in our multivariate composition for the action of M, and $\mathrm{Tuplefy}(w_1, \ldots, w_{j-1})$ is a word of ordinal removal sequences κ as above for each suitable j. As such the final character-wise map f is simply the map mapping an ordinal removal sequence $\mathbf{k}$ of length $|Q| - 1$ to the single element of $\mathrm{Remove}(Q, \mathbf{k})$.

Hopefully, this particular proof will shed some light on the multivariate composition we are using to determine $M^{\mathrm{Trunc}}(w)$: why it is shaped the way it is shaped, and what each piece of the composition is keeping track of. Utilizing the Krohn-Rhodes Theorem, we currently have the following set of generators for the regular functions:

Theorem 4. *Any regular function can be written as a multivariable composition of:*

- *Truncated Permutation-Reset Transparent Moore Machine maps,*
- *Truncated Reverse Moore Machine maps,*
- *Character-wise maps,*
- Tuplefy *(allowing us to generate multiary character-wise maps),*
- S_a *for various* a*,*
- Unpad.

It may seem like we haven't gained much, but in the next section we will see that there isn't actually that much to Permutation-Reset Transparent Moore Machines. Then we will handle the reverse Moore Machine case.

2.6 A Further Breaking Down

In this section we prove that truncated permutation-reset transparent Moore Machine maps can be written as the composition of a single truncated permutation transparent Moore Machine map and a single truncated reset transparent Moore Machine map. Then we see that permutation transparent Moore Machines and reset transparent Moore Machines are actually quite familiar objects. As before, these proofs are adapted from [5], which uses vastly different notation.

Lemma 7. *Given a permutation-reset transparent Moore Machine $M = (\Sigma, Q, q_0, \delta)$, there is a permutation transparent Moore Machine $\tilde{M}$, reset transparent Moore Machine $\vec{M}$, and function F such that:*

$$M^{\text{Trunc}}(w) = \text{Cw}_F(\vec{M}^{\text{Trunc}}(\text{Tuplefy}(w, \tilde{M}^{\text{Trunc}}(w))), \tilde{M}^{\text{Trunc}}(w))$$

Proof. Let $\tilde{M} = (\Sigma, S_Q, id, \tilde{\delta})$ and $\vec{M} = (\Sigma \times S_Q, Q, q_0, \vec{\delta})$, where S_Q is the set of all permutations on Q, and:

If δ_a is a permutation of the states of M:

$$\tilde{\delta}(f, a) = \delta_a \circ f,$$

$$\vec{\delta}(q, (a, f)) = q.$$

If δ_a is a reset on the states of M with image $\{q_a\}$:

$$\tilde{\delta}(f, a) = f,$$

$$\vec{\delta}(q, (a, f)) = f^{-1}(q_a).$$

As desired $\tilde{M}$ is a permutation automaton and $\vec{M}$ is a reset automaton (notice that the identity action is necessary in case δ_a is a permutation).

Let $F : S_Q \times Q \to Q$ with $F(f, q) = f(q)$. I now claim that

$$M^{\text{Trunc}}(w) = \text{Cw}_F(\vec{M}^{\text{Trunc}}(\text{Tuplefy}(w, \tilde{M}^{\text{Trunc}}(w))), \tilde{M}^{\text{Trunc}}(w)),$$

As desired. This is easy to verify in terms of their transition relations. The intuition behind this construction is that $\tilde{M}$ keeps track of the action of each of the permutations and $\vec{M}$ handles resets by storing them in terms of what state one would have to start in such that after being acted on by just the permutations one winds up in the current state of M. □

We now define a couple of transparent Moore Machines in order to refine our decomposition further.

Definition. *For each n, define the* Accumulator on S_n *transparent Moore Machine* AS_n*:*

$$AS_n = (S_n, S_n, id, \delta),$$

Where:

$$\delta_g(h) = h \cdot g,$$

Where S_n is the symmetric group on n elements with composition operation $(h \cdot g)(i) = h(g(i))$.

Definition. *Define the* bit-storage *automaton:*

$$Bit = (\{-, 0, 1\}, \{0, 1\}, 0, \delta),$$

Where δ_- acts as the identity, δ_0 is a reset to state 0, and δ_1 is a reset to state 1.

Since every collection of permutations can be viewed as a subset of the symmetric group S_n for some n:

Proposition 9. *Every truncated permutation transparent Moore Machine map M^{Trunc} can be written as:*

$$M^{\text{Trunc}}(w) = \text{Cw}_f(AS_n^{\text{Trunc}}(\text{Cw}_g(w))),$$

For some functions f and g.

What's more:

Proposition 10. *Every truncated reset transparent Moore Machine map M^{Trunc} can be written as:*

$$\text{Cw}_f(Bit^{\text{Trunc}}(\text{Cw}_{g_0}(w)), \ldots, Bit^{\text{Trunc}}(\text{Cw}_{g_{n-1}}(w))),$$

For suitable n, f, and $g_0, \ldots, g_{n-1}$.

Proof. Suppose $M = (\Sigma, Q, q_0, \delta)$. Choose an n such that $2^n \geq |Q|$. For every state $q \in Q$, associate a unique bitstring $b(q) \in 2^n$, where $b_k(q)$ is the bit in position k of $b(q)$, such that the start state $q_0 \in Q$ is given by the all 0s bitstring. Let $f = b^{-1}$. Suppose δ_a acts as a reset to the state q_a. Then let $g_k(a) = b_k(q_a)$. $\square$

As such:

Proposition 11. *Every truncated Moore Machine map M^{Trunc} can be written as a multivariable composition of:*

- AS_n^{Trunc} *for various n,*
- Bit^{Trunc}*,*
- *Character-wise maps.*

Now we have a much smaller generating set for our regular functions.

Theorem 5. *Any regular function can be written as a multivariable composition of:*

- AS_n^{Trunc} *for various n,*
- Bit^{Trunc}*,*
- *Truncated Reverse Moore Machine maps,*
- *Character-wise maps,*
- Tuplefy *(allowing us to generate multiary character-wise maps),*
- S_a *for various a,*
- Unpad.

2.7 Reverse Moore Machines

Just as in previous sections, we broke down the truncated Moore Machine maps into compositions involving the accumulator on S_n, the Bit automaton, and character-wise maps (note that uses of $\mathrm{Tuplefy}$ are length-preserving, and thus actually character-wise applications of a tuple-construction map), in this section, we break down truncated reverse Moore Machine maps similarly. To save ourselves work, we will simply introduce a reversal map Rev (which is not regular) to connect truncated reverse Moore Machine maps and truncated Moore Machine Maps.

Definition. *Given a word $w \in \Sigma^*$ define $\mathrm{Rev}(w)$ to be the reversal of w.*

Given a Moore Machine $M = (\Sigma, Q, q_0, \Gamma, \delta, \epsilon)$, define its reversal:

$$\mathrm{Rev}(M) = (\Sigma, Q, q_0, \Gamma, \delta, \epsilon).$$

And similarly define the reversal of a reverse Moore Machine.

Proposition 12. *Given a Moore Machine M, and word w:*

$$\mathrm{Rev}(M)(w) = \mathrm{Rev}(M(\mathrm{Rev}(w))).$$

What's more:

$$\mathrm{Rev}(M)^{\mathrm{Trunc}}(w) = \mathrm{Rev}(M^{\mathrm{Rest}}(\mathrm{Rev}(w))).$$

Functions that are related in this way we say are related by *conjugation by* Rev. This relation is reflexive and symmetric. What's more since Rev is its own inverse, if f and f' are related by conjugation and g and g' are related by conjugation, then $f \circ g$ and $f' \circ g'$ will be related by conjugation. Indeed this works for multiary functions as well:

Definition. *Given an n-ary function f, we say that*

$$(x_0, \ldots, x_{n-1}) \mapsto f(x_0, \ldots, x_{n-1})$$

And

$$(x_0, \ldots, x_{n-1}) \mapsto \mathrm{Rev}(f(\mathrm{Rev}(x_0), \ldots, \mathrm{Rev}(x_{n-1})))$$

Are related by conjugation by Rev.

Proposition 13. *Given a multi-ary composition of functions, if you replace each function by its conjugation by Rev, the overall composition is related to the original composition by conjugation by Rev.*

Proof. It suffices to note that in the resulting composition, whenever the output of a function is fed into the input of another function, it is reversed twice, effectively doing nothing to it. □

Additionally, conjugation by Rev does not alter character-wise functions.

Finally, we prove a final breakdown:

Proposition 14. *Given a transparent Moore Machine $M = (\Sigma, Q, q_0, \delta)$, we can write M^{Rest} as a composition of character-wise maps and M^{Trunc}.*

Proof. It is easy to verify that:

$$M^{\mathrm{Rest}}(w) = \mathrm{Cw}_\delta(M^{\mathrm{Trunc}}(w), w).$$

□

It follows from this that for any Moore Machine M, M^{Rest} can be written as a composition of character-wise maps and M^{Trunc}.

Proposition 15. *Any truncated reverse Moore Machine map can be written as the multivariate composition of:*

- RAS_n^{Trunc}*, where* RAS_n *is the reversal of* AS_n*, for various* n*,*
- $RBit^{\text{Trunc}}$*, where* $RBit$ *is the reversal of* Bit*,*
- *Character-wise maps.*

Proof. Every reverse Moore Machine is $\mathrm{Rev}(M)$ for some M. As such, we can write $\mathrm{Rev}(M)^{\text{Trunc}}$ as:

$$\mathrm{Rev} \circ M^{\text{Rest}} \circ \mathrm{Rev}.$$

As we've seen, we can write M^{Rest} as a multiary composition of AS_n^{Trunc} for various n, Bit^{Trunc}, and character-wise maps, so by our conjugation of compositions lemma, we can write $\mathrm{Rev}(M)^{\text{Trunc}}$ as a multiary composition of the conjugations of those components, which is what we were trying to prove. $\square$

While we're at it, note that:

Proposition 16. *Given a Moore Machine* M*, we can write* $M(w)$ *as the multivariable composition of a truncated Moore Machine map and* $S_{\#}$*.*

Proof. Augment M to M' by allowing it to interpret the input # (it may do so in any way it likes). Then:

$$M(w) = M^{\text{Trunc}}(S_{\#}(w)).$$

$\square$

As one final refinement of our generating set, we show that RAS_n is unnecessary as a generator.

2.8 Removing RAS_n

Note that AS_n and RAS_n are very similar automata. For AS_n, one interprets the input character $w[i]$ as a permutation relating $AS_n(w)[i]$ and $AS_n(w)[i+1]$. For RAS_n, one interprets the input character $w[i]$ as a permutation relating $RAS_n(w)[i+1]$ and $RAS_n(w)[i]$. Since every permutation has an inverse, shouldn't these two automata be the same up to a suitable character-wise map on the inputs? Certainly not! For there is another constraint on the runs of AS_n and RAS_n we must account for. For AS_n, the first character of its run is specified to be id. For RAS_n, the last character of its run is specified to be id.

Let's compare $AS_n(w)$ and $RAS_n(\mathrm{Cw}_{inverse}(w))$ on some generic five character input $w = abcde$:

w	a	b	c	d	e	
$AS_n(w)$	id	a	ab	abc	$abcd$	$abcde$
$RAS_n(\mathrm{Cw}_{inverse}(w))$	$(abcde)^{-1}$	$(bcde)^{-1}$	$(cde)^{-1}$	$(de)^{-1}$	e^{-1}	id

Table 2.2: Comparing AS_n and RAS_n on Sample Input

In addition to applying a suitable transformation to the inputs of AS_n, we must also apply a suitable transformation to the outputs of AS_n if we want to produce the output of RAS_n. Specifically, if we multiply every character in the output of AS_n on the left by the inverse of the last character of the output, it will ensure that the new last character of the output is id, but still maintain the transition relationships. This requires passing the information of the last character of AS_n to every other position.

First, we must introduce the *mask* of our word w, a word $\mathrm{Mask}(w)$ which is all 0s up to the length of w, followed by a 1. This is computed simply by taking $S_1(\mathrm{Cw}_0(w))$ where 0 is the constant 0 map. Despite its simplicity, this word will be key to performing our computation.

Consider the reverse reset Transparent Moore Machine $R = (\{0,1\} \times S_n, S_n, id, \delta)$ with:

$$\delta(q,a) = \begin{cases} b & a = (1,b) \\ q & a = (0,b) \end{cases}$$

This is a reverse reset transparent Moore Machine, and so R^{Trunc} can be written in terms of character-wise maps and $RBit^{\mathrm{Trunc}}$.

Consider the action of R^{Trunc} on $\mathrm{Cw}_{pair}(\mathrm{Mask}(w), AS_n(w))$:

w	a	b	c	d	e	
$\mathrm{Mask}(w)$	0	0	0	0	0	1
$AS_n(w)$	id	a	ab	abc	$abcd$	$abcde$
$R^{\mathrm{Trunc}}(\mathrm{Cw}_{pair}($“$))$	$abcde$	$abcde$	$abcde$	$abcde$	$abcde$	$abcde$

Table 2.3: The Action of R^{Trunc} on $\mathrm{Cw}_{pair}(\mathrm{Mask}(w), AS_n(w))$

We now have a word which we can combine with $AS_n(w)$ via the appropriate bitwise map (multiplying by the inverse on the left) to produce $RAS_n(\mathrm{Cw}_{inverse}(w))$.

However, what we wanted was $RAS_n^{\mathrm{Trunc}}(\mathrm{Cw}_{inverse}(w))$. We can attain that by combining $RAS_n(\mathrm{Cw}_{inverse}(w))$ with $\mathrm{Mask}(w)$ bitwise to replace the last character of $RAS_n(\mathrm{Cw}_{inverse}(w))$ with a # and then using Unpad to remove it.

Letting f map pairs of the form $(0,a)$ to a and pairs of the form $(1,a)$ to #:

w	a	b	c	d	e	
$\mathrm{Mask}(w)$	0	0	0	0	0	1
$RAS_n(\mathrm{Cw}_{inverse}(w))$	$(abcde)^{-1}$	$(bcde)^{-1}$	$(cde)^{-1}$	$(de)^{-1}$	e^{-1}	id
$\mathrm{Cw}_f(``)$	$(abcde)^{-1}$	$(bcde)^{-1}$	$(cde)^{-1}$	$(de)^{-1}$	e^{-1}	$\#$
$\mathrm{Unpad}(``)$	$(abcde)^{-1}$	$(bcde)^{-1}$	$(cde)^{-1}$	$(de)^{-1}$	e^{-1}	

Table 2.4: Producing RAS_n^{Trunc} on Sample Input

This composition produces the desired output and works in general. As such, we have the following, final form, of our theorem:

Theorem 6. *Any regular function can be written as a multivariable composition of:*

- AS_n^{Trunc} *for various* n,
- Bit^{Trunc},
- $RBit^{\mathrm{Trunc}}$,
- *Character-wise maps,*
- Tuplefy,
- S_a *for various* a,
- Unpad.

2.9 Further Research

We may think of a regular function as a function whose output can be verified to be correct by a deterministic finite automaton, and in this chapter we have spelled out exactly how one might go about finding that appropriate output.

This should feel very similar to the class of NP problems or recursive enumerability, in that once one has an answer one can verify it quickly. In these more advanced cases, the issue of finding that answer is significantly more difficult. In the case of finite automata, one can get by mostly with just a two-pass system.

While thinking about automata as transducers instead of acceptors takes us away from the underlying logic, it takes us closer to real-world applications of automata. One is then lead to ask similar questions about functions whose graphs are recognized by Büchi automata, Tree automata, and Rabin automata. The determinization-harvester decomposition can be adapted to trees, but is there an analog to the Krohn-Rhodes theorem for trees in this context? What about in the case of Büchi automata or Rabin automata, for which there is no end of the input to start the harvester running backwards from? Can nice generators still be found?

There is still much work to be done in establishing a Büchi-Elgot-Trakhtenbrot theorem for graphs. There are several nice candidates for a monadic second-order logic of graphs, and some nice notions of automata operating on graphs, but no full correspondence between them. The current state of the art is Courcelle's Theorem (see, e.g. [2]), which allows us to translate questions in a graph logic to tree automata operating on a tree decomposition of the original graph, but not back. Fortunately, this is the direction most of interest to applications. But perhaps an approach which instead of trying to connect formulas and acceptors, connected describable functions and transducers would shed light on the problem?

Finally, we also note that it is not possible to break down our AS_n generators much further. Of course for large enough n one may decompose the symmetric group S_n as a semidirect product of the alternating group A_n and S_2, and use this decomposition to guide a slight decomposition of AS_n, but this doesn't gain anything. It is likely no further decomposition is possible. One would then like a proof that, for instance, accumulators on the cyclic groups do not suffice, in a way that hopefully sheds some light on what behavior symmetric groups capture that cyclic groups cannot. Alternately, a decomposition of the accumulators on the symmetric groups in terms of accumulators on the cyclic groups would be a remarkable result.

CHAPTER 3

AXIOMATIZING THE WEAK SECOND ORDER THEORY OF ONE SUCCESSOR

In this chapter we present an axiomatization of the theory of the Weak Second Order Theory of One Successor (WS1S), and show that it is complete. Furthermore, we will prove that no such axiomatization can be finite. This is in preparation for chapter 4 which will explore the nonstandard models of WS1S.

3.1 Introduction

The *weak second order theory of one successor*, WS1S, is the theory of the two-typed structure $G := (\omega, \mathcal{F}(\omega), \in, S)$, where $\mathcal{F}(\omega)$ is the set of finite subsets of ω, $\in$ is the binary relation between types (membership), and S is the successor operation on ω. We will use lower case variables a, b, ℓ, x, y, z to denote elements of the first type, and upper case variables $A, B, X, Y, Z, \dots$ to represent elements of the second type. Boldface variables $\mathbf{x}, \mathbf{y}, \mathbf{X}, \mathbf{Y}$ will indicate shorthand for finite or infinite sequences of elements of the corresponding type indexed by natural numbers. This structure is particularly interesting not only for being an actually decidable second order theory of ω (the addition of $+$ or weak dyadic second order types makes it undecidable), but also for the correspondence between its theory and the theory of finite state automata which allows it to be decidable in the first place.

Previously an axiomatization of the Weak Second Order Theory of Two Successors (WS2S), whose techniques and results can be reduced to the WS1S case, was completed by Siefkes [10]. However this axiomatization focused on inductive axiom schema, suggesting an analogy between WS1S and, e.g. Peano Arithmetic. This paper provides an axiomatization that stresses the weakness of WS1S and the interconnection between WS1S and finite automata.

As it happens, a finite axiomatization is too much to hope for, but we will be able to provide an axiomatization with simple statements that are self-evident of G, along with a schema of axioms which can be interpreted as asserting that finite automata of a certain class have a run on any input. These schema will be fairly complex and only self-evident in the presence of this interpretation. We will not aim in this paper to have our axiomatization be minimal. Instead, the axiomatization presented in this chapter has been chosen specifically to motivate and set the foundation for chapter 4, investigating the nonstandard models of WS1S.

Closely resembling the approach in [10], our approach to proving the completeness of our axiomatization is as follows: given a formula ϕ in the language of WS1S, we will inductively define a finite automaton A_ϕ which is expected to be equivalent to the formula ϕ[1]. And given a finite automaton A, we will define a formula Φ_A which is expected to be equivalent to the automaton A[2]. This will serve as a canonical form for formulas in WS1S. We will then prove that our axioms suffice to show that every formula is equivalent to its canonical form. These steps will resemble the proof of the Büchi-Elgot-Trakhtenbrot Theorem, the equivalence of formulas in WS1S and finite automata, with every step being shown to follow from our presented axioms. Finally, we will prove that every true sentence in canonical form can be proved from our axioms.

[1]That is, A_ϕ will accept a suitable encoding of $X_0, \ldots, X_{n-1}$ if and only if $\phi(X_0, \ldots, X_{n-1})$.

[2]That is, $\Phi_A(X_0, \ldots, X_{n-1})$ will hold if and only if A accepts a suitable encoding of $X_0, \ldots, X_{n-1}$.

Proving that WS1S cannot be finitely axiomatized will then be a fairly straightforward matter. Suppose we have a finite complete axiomatization, which we can by taking a conjunction reduce to a single axiom ϕ. This formula will follow from our axiomatization, and thus follow from some finite subset of our axiomatization. We will then produce a model H of this subset of our axiomatization which does not satisfy all of our axioms. ϕ will be true of H, contradicting our assumption that ϕ was a complete axiomatization (and thus only true of models that satisfied the entire theory of WS1S).

3.2 Axiomatizing $(\omega, <)$

One might wonder how the first and second order parts of WS1S interact with each other. More formally, are there relations on first order elements which require the use of second order quantification in their definition? One such relation is the ordering $<$ on ω.[3] It will be convenient to refer to it in our axiomatization, and each of its appearances will stand in as shorthand for the definition below:

Definition. *A set X is said to be* downward closed *if:*

$$\mathrm{dc}(A) :\equiv \forall X : S(x) \in A \implies x \in A$$

We define a linear ordering on the first order elements by:

$$x \leq y :\equiv \forall X : \mathrm{dc}(X) \wedge y \in X \implies x \in X$$

As such it will be useful to throw in a complete axiomatization of $(\omega, <)$ to our axiom system. One such axiomatization is provided in Chapter 13 of [9],

[3]Another is modular equivalence for any fixed modulus.

which is adapted below to use the already-present notation for the successor operation.

Axiom 1. *$<$ forms a Linear Ordering with Smallest Element and with Successor Operation S which is All But Onto. That is:*

(1.1) $\forall x, y, z : x < y \wedge x < z \implies x < z$ *($<$ is transitive)*

(1.2) $\forall x, y : x < y \implies \neg(y < x)$ *($<$ is antisymmetric)*

(1.3) $\forall x, y : x < y \vee x = y \vee y < x$ *($<$ is total)*

(1.4) $\forall x : S(x) > x$ *(S is strictly increasing)*

(1.5) $\forall x : \nexists y : x < y < S(x)$ *(S is a successor operation)*

(1.6) $\exists o : \forall x : o \leq x$ *($<$ has minimal element)*

(1.7) $\exists o : \forall x : x \neq o \implies \exists y : S(y) = x$ *(S has image $\omega \setminus \{o\}$)*

Note that it follows from **(1.1)** and **(1.7)** that the o in **(1.6)** and **(1.7)** are unique, and from **(1.4)** that the o in **(1.6)** and **(1.7)** are the same. We denote this element with the shorthand 0.

It is worth noting that this axiomatization does not define $(\omega, <)$ uniquely. It is simply a base from which all other first order facts about $(\omega, <)$ can be proved. As mentioned in [9], this set of axioms characterizes the linear orderings of the form ω followed by $\mathbb{Z} \times L$ ordered lexicographically ($\omega + \zeta \cdot L$), for some linear ordering L. These linear orderings all have the same first order theory.

Together, all of these axioms set up the foundation upon which we will develop the second order part of our axiomatization.

3.3 Basic Set Axioms

In this section, we introduce some basic axioms for the second order elements of WS1S. These first few have been intentionally chosen to resemble axioms of ZFC. We will also define a notion of *bitwise operation* and show that our axioms suffice to show that the second order elements of WS1S are closed under all the bitwise operations we could expect them to be closed under.

Axiom (Extensionality). *Sets are equal if and only if they contain the same elements.*

Axiom (Singleton). *For any x, there is a set, denoted $\{x\}$, containing only x.*

Definition. *Given two sets A, B, we write:*

$$C = A \cup B :\equiv \forall x : x \in C \iff x \in A \vee x \in B$$

$$C = A \cap B :\equiv \forall x : x \in C \iff x \in A \wedge x \in B$$

$$C = A \setminus B :\equiv \forall x : x \in C \iff x \in A \wedge x \notin B$$

$$A \subseteq B :\equiv \forall x : x \in A \implies x \in B$$

Axiom (Closure Under Boolean Operations). *The sets are closed under the boolean operations $\cup$ and $\setminus$.*

We can interpret second order elements of WS1S with strings of 0s and 1s via their characteristic functions, that is, given a set X, we can produce the sequence s where $s_n = 1$ iff $n \in X$. Via this interpretation, we can define a notion of *bitwise map*, where the membership of a number in the output depends only on which of the inputs it is a member of. Obviously the set of finite sets of natural numbers is not closed under all bitwise maps. Bitwise negation, that is, complementation, takes a finite set to a cofinite set. However, this is essentially the only obstruction.

Definition. *We say that an n-ary boolean function $f : \{0,1\}^n \to \{0,1\}$ is* bounded *if it takes $(0,\dots,0)$ to 0.*

Definition. *Given an n-ary boolean function $f : \{0,1\}^n \to \{0,1\}$, we define the* bitwise-f *map* Bw_f*:*

$$\mathrm{Bw}_f : (\mathcal{P}(X))^n \to \mathcal{P}(X),$$

Such that:

$$k \in \mathrm{Bw}_f(X_0,\dots,X_{n-1}) \iff f(\delta_{k\in X_0},\dots,\delta_{k\in X_{n-1}}) = 1,$$

Where of course:

$$\delta_{k\in X} = \begin{cases} 0 & k \notin X, \\ 1 & k \in X. \end{cases}$$

Definition. *Given an indexed collection $\mathbf{X}$ of sets $X_0,\dots,X_{n-1}$ we denote by $\mathbf{X}[k]$ the set of positions i such that $k \in X_i$:*

$$(\mathbf{X}[k])_i := \{i | k \in X_i\}.$$

Given a tuple $\mathbf{s} \in \{0,1\}^n$ and indexed collection of sets $X_0,\dots,X_n$, we denote by $X_{\mathbf{s}}$ the set of positions k where the set of indices i such that $k \in X_i$ is exactly the positions of 1s in $\mathbf{s}$:

$$X_{\mathbf{s}} := \{k | k \in X_i \iff s_i = 1\}.$$

Lemma 8. *Given a tuple $\mathbf{s} \in \{0,1\}^n$, if $\mathbf{s} \neq (0,\dots,0)$, then $X_{\mathbf{s}}$ can be written as a combination of $X_0,\dots,X_{n-1}$ using only $\cup$ and $\setminus$.*

Proof. First, we note that intersection, $\cap$, can be written using only $\setminus$:

$$A \cap B = A \setminus (A \setminus B).$$

From this, we can use associativity to write arbitrary intersections in terms of $\cup$ and $\setminus$.

Next, note that for any s $\in \{0,1\}^n$,

$$\bigcup_{j|s_j=1} X_j$$

Is the set of positions k where $\mathbf{X}[k]$ contains as a subset all the positions of 1s in s.

Next, note that

$$\bigcup_i X_i$$

Is the set of positions k where $\mathbf{X}[k]$ is nonempty.

Next, note that for any j,

$$(\bigcup_i X_i) \setminus X_j$$

Is the set of positions k where $\mathbf{X}[k]$ is nonempty and does not contain j.

As such, for any s $\in \{0,1\}^n$ (with s $\neq (0,\ldots,0)$),

$$\bigcap_{j|s_j=0} ((\bigcup_i X_i) \setminus X_j)$$

Is the set of positions k where $\mathbf{X}[k]$ is nonempty and does not contain any positions of 0s in s.

Thus it is that

$$X_{\mathbf{s}} = \left(\bigcup_{j|s_j=1} X_j\right) \cap \left(\bigcap_{j|s_j=0} ((\bigcup_i X_i) \setminus X_j)\right).$$

□

Proposition 17. *Any bitwise bounded boolean function can be written as a combination of* $\cup$ *and* $\setminus$*.*

Proof. We can simply write:

$$\mathrm{Bw}_f(X_0, \ldots, X_{n-1}) = \bigcup_{\mathbf{s} \mid f(\mathbf{s})=1} X_{\mathbf{s}}.$$

□

As such, closure under union and difference is enough make sure that the second order objects are closed under all bitwise bounded boolean functions.

Finally, we provide an axiom to ensure that all of the second order objects are bounded. There's a subtlety here: not even our monadic second order logic can say directly that every set has (arbitrarily) finitely many elements, but it is possible to say that every set has an upper bound, which in $(\omega, <)$, is equivalent.

Axiom. *Every nonempty second order element has a $<$-maximal element. We denote the maximal element of a set X by* $\max(X)$

We note that it follows from this axiom and the S is a successor axiom that every set has a downwards closure.

Definition. *Given a set X, we denote by its downward closure, dcl(X), the $\subseteq$-least downwards closed set containing X as a subset.*

Proposition 18. *It follows from our axioms that every set has a downwards closure.*

Proof. Given a set X, X has a $<$-maximal element, denote it by x. Since $S(x) > x$, there must be a downwards closed set Y containing x which does not contain x (otherwise we would have $x \geq S(x)$. I claim that Y contains all of X, since any other element $z \in X$ satisfies $z < x$, and thus since Y contains x and is downwards closed, this means Y contains z as well by the definition of $<$.

I also claim that Y is $\subseteq$-minimal, in that any other downwards closed set containing x must also contain something strictly larger than x. Suppose we have two downwards closed sets containing x, $W \neq W'$. Without loss of generality, we may assume there is some $w \in W$ with $w \notin W'$. Since w is not in W', a downwards closed set containing x. Thus we have that $w \nleq x$, and so $w > x$ (by totality). So there cannot be two downwards closed sets containing x as their maximal element. $\square$

Finally, we note that atoms under $\subseteq$ are actually singletons.

Proposition 19 (No spurious singletons)**.** *A nonempty set X contains only itself and the emptyset as subsets iff it is a singleton $\{x\}$ for some x.*

Proof. ($\Rightarrow$): Suppose we have a nonempty set X containing only itself and the emptyset as subsets. Since X is nonempty, it contains an element, say x. Suppose X also contains an element $y \neq x$. Then $X \setminus \{x\}$ is neither X nor empty, contradicting our hypothesis. So $X = \{x\}$.

($\Leftarrow$): Suppose $Y \subseteq \{x\}$ and Y is nonempty. Then Y must contain an element y. However, in order to be a subset, $y \in \{x\}$, hence $y = x$. Y cannot contain any other elements, as $\{x\}$ doesn't contain any other elements. $\square$

A simple result, but a critical one later, since we will need to represent first order elements as singletons, and to do that we will need to be certain that singletons are describable in terms of their $\subseteq$ behaviour alone.

We summarize and number these axioms as follows:

Axiom 2. *The second order objects are defined by their elements, closed under bounded bitwise operations, closed under finite modification, and bounded.*

(2.1) $\forall X, Y : (\forall x : x \in X \Leftrightarrow x \in Y) \Rightarrow X = Y$ *(Extensionality)*

(2.2) $\forall x : \exists X : \forall y : y \in X \Leftrightarrow y = x$ *(Singleton)*

(2.3) $\forall X, Y : \exists Z : \forall x : x \in Z \Leftrightarrow (x \in X \vee x \in Y)$ *(Union Closure)*

(2.4) $\forall X, Y : \exists Z : \forall x : x \in Z \Leftrightarrow (x \in X \wedge x \notin Y)$ *(Difference Closure)*

(2.5) $\forall X : \exists x : x \in X \wedge \forall y : y \in X \Rightarrow y \leq x$ *(Bounding)*

3.4 Logical Automata

In this section we will introduce a notion of *logical automata*, automata which have been modified to operate within WS1S. Alternately, we can think of this in terms of canonical forms for formula. WS1S has a very nice canonical form for formulas, and logical automata will provide a semantically transparent parameterization of these canonical forms.

Recall that if $\mathbf{X}$ is an indexed collection with finite set of indices K, and s $\in \{0,1\}^K$, then we use the shorthand $X_\mathbf{s}$ to stand for $\{x | \forall k \in K : \mathbf{s}_k = 1 \iff x \in X_k\}$. Thus, we write $x \in X_\mathbf{s}$ as shorthand for:

$$x \in X_\mathbf{s} :\equiv \left[\bigwedge_{k|\mathbf{s}_k=1} x \in X_k\right] \wedge \left[\bigwedge_{k|\mathbf{s}_k=0} x \notin X_k\right].$$

We also use the shorthand n for the set $\{0, \ldots, n-1\}$, but never in reference to the second order element of a model of WS1S.

Definition. *A* logical automaton *is a 5-tuple $A = (n, Q, I, \delta, F)$ where:*

- *n is the number of inputs, potentially 0.*
- *Q is a finite set of states. We will refer to the elements of Q via some canonical ordering as $0, \ldots, |Q|-1$, but often the states will retain additional information,*

for instance they may be pairs of states of other automata. We will also refer to the number of states of an automaton A as $|A|$. Often we will prove results for automata with state set m where this suffices to prove the result in general.

- $I \subseteq Q$ *is the set of* initial states.
- $\delta \subseteq Q \times 2^n \times Q$ *is the* transition relation.
- $F \subseteq Q$ *is the set of* final states.

These automata are specifically designed to accept n-tuples of second order elements of a model of WS1S. Given an n-tuple of second order objects $\mathbf{X} = (X_0, \ldots, X_{n-1})$, we say that a first order-object ℓ and a $|Q|$-tuple of second order objects $\mathbf{Y} = (Y_0, \ldots, Y_{|Q|-1})$ is a run *of A on input $\mathbf{X}$ if:*

- *ℓ is strictly larger than anything in any X_i (particularly, the least strict upper bound to the X_i).*
- *Y_i form a partition of dcl(ℓ), that is, they have pairwise empty intersection and together their union is all of dcl(ℓ).*
- *$0 \in Y_q$ for some $q \in I$.*
- *For any $0 < x < \ell$, if $x \in Y_q$ and $x \in X_{\mathbf{s}}$ and $S(x) \in Y_{q'}$ then $(q, \mathbf{s}, q') \in \delta$.*

We say that such a run is accepting *if additionally:*

- *$\ell \in Y_q$ for some $q \in F$.*

We interpret the definition of a run as follows: first order elements represent instances of time. If $x \in Y_q$ for some state q (and since the Y_q form a partition of dcl$(\{\ell\})$, this happens for exactly one state provided $x \leq \ell$) we interpret this as the automaton being in state q at time x. Of course, we require the automaton to be in exactly one state at any time, and we require the automaton to be in a start state at time 0. Then we require that if the automaton is in a state q at time x and it reads input s (the input being read at any one time is just a binary sequence representing whether x is in each of the inputs.) at time x, then at time $S(x)$, the automaton should be in a state q' such that q, s, and q' are related by the transition relation. We also require that the automaton stop at some time ℓ, after having read all the input. Note that at time $\max(\bigcup_{i\in n} X_i)$, the state of the automaton has not yet been updated to take into account the input at time $\max(\bigcup_{i\in n} X_i)$ which by construction is nontrivial, so we require that ℓ be strictly larger than this.

A design choice has been made as to whether the state at time x should be the state of the automaton before reading the input at time x or after. There are advantages and disadvantages to both, and both give equivalent notions of automata, but it seems like the former option is preferable, due to saving steps on many inductive proofs throughout this chapter.

Next, given an automaton $A = (n, Q, I, \delta, F)$, we construct a formula $\Phi_A(X_0, \dots, X_{n-1})$ which formalizes the above.

Definition. *Define the formula* $R_{(n,Q,I,\delta,F)}(X_0, \dots, X_{n-1}, \ell, Y_0, \dots, Y_{|Q|-1})$ *in WS1S as the conjunction of:*

- $\forall x : \bigvee_{i\in Q} \bigwedge_{j\in Q}[x \in Y_j \iff (i = j \wedge 0 \leq x \leq \ell)]$
- $\bigvee_{i\in I} 0 \in Y_i$
- $\forall x : 0 \leq x < \ell \implies \left[\bigvee_{(i,\mathbf{s},k)\in\delta}(x \in Y_i \wedge x \in X_{\mathbf{s}} \wedge S(x) \in Y_k)\right]$

This formalizes the notion of $(\ell, Y_0, \ldots, Y_{|Q|-1})$ being a run of (n, Q, I, δ, F) on input $X_0, \ldots, X_{n-1}$.

Definition. *Define the formula $\Phi_{(n,Q,I,\delta,F)}(X_0, \ldots, X_{n-1})$ in WS1S as:*

$$\begin{aligned}\exists \ell : \exists Y_0, \ldots, Y_{|Q|-1} : \ell = \mathrm{LSUB}(\bigcup \mathbf{X})& \\ \wedge R_{(n,Q,I,\delta,F)}(X_0, \ldots, X_{n-1}, \ell, Y_0, \ldots, Y_{|Q|-1})& \\ \wedge \bigvee_{i \in F} \ell \in Y_i.&\end{aligned}$$

Where $\mathrm{LSUB}(X)$ is shorthand for the least strict upper bound for a set, either 0 for the emptyset or $S(\max(X))$ for a nonempty set.

This formalizes the notion of there being an accepting run of (n, Q, I, δ, F) on input $X_0, \ldots, X_{n-1}$.

We are now prepared to introduce our axiom schema for finite automata. It suffices in this case to simply specify that automata have runs on any input. We will be able to prove later theorems involving whether automata have accepting runs or not. Of course, not every automaton has a run on any input, but there are simple things we can check to ensure that every automaton has a run on any input.

Definition. *An automaton (n, Q, I, δ, F) is said to be* runnable *if:*

- $|I| > 0$,
- $\forall q, \mathrm{s} \in 2^n : \exists q' : \delta(q, \mathrm{s}, q')$.

Definition. *An automaton (n, Q, I, δ, F) is said to be* reverse-runnable *if:*

- $|F| > 0$,

- $\forall q, \mathbf{s} \in 2^n : \exists q' : \delta(q', \mathbf{s}, q)$.

Note that these are assertions about the internal structure of the automaton: nothing about the runs or acceptance of the automaton. As such, the following axiom schema is nontrivial:

Axiom. (3.A) *If $A = (n, Q, I, \delta, F)$ is a runnable automaton, for any sequence of inputs $X_0, \ldots, X_{n-1}$,*

$$\Phi_{(n,Q,I,\delta,Q)}(X_0, \ldots, X_{n-1}).$$

Axiom. (4.A) *If $A = (n, Q, I, \delta, F)$ is a reverse-runnable automaton, for any sequence of inputs $X_0, \ldots, X_{n-1}$,*

$$\Phi_{(n,Q,Q,\delta,F)}(X_0, \ldots, X_{n-1}).$$

This is a very specific sort of inductive axiom: if we can start constructing a run of our automaton and at any point if we have a partial run, we can find a state to transition to next, then we can construct the whole run of our automaton. In the runnable case we construct our run from start to end, and in the reverse-runnable case, we construct a run from end to start. Note that in the runnable case, we don't have to worry about our run being accepting, and in the reverse-runnable case, we don't have to worry about our run having the right start state, since we've replaced F and I respectively with all of Q. This completes our axiomatization.

3.5 Basic Consequences of Our Axioms

In this section we collect some of the basic consequences of our axioms, in preparation for future sections.

First, we define a *predecessor operation* P which is meant as an inverse to S, and define the operations S'', P'' on second order elements which are meant to apply their respective operations to each element in the set.

Definition. *Define*

$$P(x) := \begin{cases} 0 & x = 0, \\ \\ y & x = S(y). \end{cases}$$

Given a second order element X, define $S''(X)$ and $P''(X)$ such that:

$$S(x) \in S''(X) \iff x \in X,$$

and $0 \notin S''(X)$, and:

$$x \in P''(X) \iff S(x) \in X.$$

Proposition 20. *It follows from our axioms that the second order objects are closed under S''.*

Proof. Construct the automaton

$$B := (1, \{0, 1\}, \{0\}, \delta, \{0, 1\}),$$

Where:

$$\delta = \{(i, j, j) | i, j \in \{0, 1\}\}.$$

Let $\mathbf{Y}$ be a run of B on input X. Then it is easy to check that Y_1 is $S''(X)$ by looking at the transition relation. □

Proposition 21. *It follows from our axioms that the second order objects are closed under P''.*

Proof. Construct the automaton

$$B = (1, \{0,1\} \times \{0,1\}, \{0,1\} \times \{0,1\}, \delta, \{(0,0)\}),$$

Where:

$$\delta = \{((i,j), k, (j,k)) | i,j,k \in \{0,1\}\}.$$

Let $\mathbf{Y}$ be a run of B on input X (this uses **(4.A)**). Then it is easy to check by looking at the transition relation that $Y_{(1,0)} \cup Y_{(1,1)}$ is just X, and from this that $Y_{(0,1)} \cup Y_{(1,1)}$ is $P''(X)$. □

Secondly, I assert that every set has a $<$-least element. This is a key result that allows us to perform inductive proofs: if we want to show that something always happens for any first order element, $\forall x : P(x)$, then it suffices to prove three things:

1. $P(0)$,
2. $P(x) \implies P(S(x))$,
3. The set of x such that $\neg P(x)$ is a second order element of our model[4].

I claim it will then follow that $\forall x : P(x)$. Let Y be the set of x such that $\neg P(x)$. Then Y will have a least element. This least element cannot be 0 or the successor of anything, contradicting **(1.7)**. Typically, we will prove (3) by application of the closure axioms **(2.2)**, **(2.3)**, **(2.4)**, **(3.A)**, and **(4.A)**.

We can even show that a particular run satisfies the transition relation of an automaton using induction. In order to use induction, we need to be able to talk about the set of positions where, given an automaton A and input $\mathbf{X}$, a proposed

[4]Or, at least, for any y, the set $\{x | x < y \wedge \neg P(x)\}$ is a second order element of our model.

run $\mathbf{Y}$ fails to satisfy the transition relation δ for A. These positions are given by:

$$\bigcup_{(a,\mathbf{s},b)\notin\delta} [Y_a \cap X_{\mathbf{s}} \cap P''(Y_b)].$$

Typically, though, there will be a more direct proof: if $\mathbf{X}, \mathbf{Y}$ are defined in such a way as to satisfy a particular transition relation, then it will usually follow directly that they satisfy the transition relation for our automaton A.

Proposition 22. *It follows from our axioms that every nonempty set has a $<$-least element.*

Proof. Begin with a set X. Consider the following automaton:

$$\mathrm{Bit} := (1, \{0,1\}, \{0\}, \delta, \{0,1\}),$$

Where:

$$\delta := \{(0,0,0), (0,1,1), (1,0,1), (1,1,1)\}.$$

Now, let ℓ, Y_0, Y_1 be the run of Bit on input X. ℓ will be the successor of the largest element of X. Y_0 will be the set of positions up to and including the first appearance of an element of the input, and Y_1 will be all positions afterwards up to and one past the last appearance of an element of the input.

First, I assert that if $z \in Y_0$ and $w < z$ then $w \in Y_0$. Suppose not. Then there is a largest w with $w < z$ and $w \in Y_1$, the largest element of $Y_1 \cap \mathrm{dcl}(\{z\})$. This w cannot be z, since $z \in Y_0$, so it must be that $S(w) \leq z$. But since w was the largest element of $Y_1 \cap \mathrm{dcl}(\{z\})$, it must be that $S(w) \in Y_0$, contradicting the construction of our transition relation.

Y_0 is nonempty since 0 is the starting state. Thus, By **(2.5)**, Y_0 has a largest element y. Since $\{Y_0, Y_1\}$ is a partition of $\mathrm{dcl}(\{\ell\})$, $S(y)$ is either in Y_1 or not in $\mathrm{dcl}(\{\ell\})$ at all, that is, $S(y) > \ell$ (by totality).

Case 1: suppose that $S(y)$ is in Y_1. Since $y \in Y_0$ and $S(y) \in Y_1$, it must be by the transition relation that $y \in X$. Furthermore, I assert that there is no $z < y$ such that $z \in X$, since otherwise by the transition relation, $S(z)$ would be in Y_1, contradicting our earlier proved downward closure of Y_0. So this y is the smallest element of X.

Case 2: suppose that $S(y) > \ell$. Since $y \in \text{dcl}(\{\ell\})$, it must be that $y \leq \ell$, thus since S is a successor operation, $y = \ell$. By the downward closure of Y_0, it must be that $Y_0 = \text{dcl}(\{\ell\})$. I claim in this case that X is empty, since if it contained an element w, then $S(w)$ would be in Y_1 and Y_1 must be empty, since $\{Y_0, Y_1\}$ is a partition of $\text{dcl}(\{\ell\})$. □

3.6 Determinization

One goal in this chapter is to prove that the standard Büchi-Elgot-Trakhtenbrot result, equivalence of finite automata and WS1S formulas, holds for our logical automata, and that this equivalence is a consequence of our axioms. Specifically, given a formula ϕ, we wish to prove that there is an automaton A such that:

$$\phi(X_0, \ldots, X_{n-1}) \iff \Phi_A(X_0, \ldots, X_{n-1}).$$

We will denote this automaton A_ϕ. As such, much of the next few sections is dedicated to reproving classical results of automata theory, from the given axioms.

Critical to the importance of finite automata is the expressive equivalence of deterministic and non-deterministic automata. While non-deterministic automata are much more natural from an atemporal, relational, formal logic stand-point, being as they are simply a specific case of a much more general class of local constraint systems[5], deterministic automata are much more natural from a temporal automaton perspective. Additionally, it is clear how to complement them, a critical problem in the theory of general local constraint systems.

Definition. *An automaton* $A = (n, Q, I, \delta, F)$ *is* deterministic *if:*

- $|I| = 1$,
- $\forall q, \mathbf{s} \in 2^n : \exists! q' : \delta(q, \mathbf{s}, q')$.

An automaton $A = (n, Q, I, \delta, F)$ *is* reverse-deterministic *if:*

- $|F| = 1$,
- $\forall q, \mathbf{s} \in 2^n : \exists! q' : \delta(q', \mathbf{s}, q)$.

Proposition 23. *It follows from our axioms that every deterministic automaton* $A = (n, Q, I, \delta, F)$ *has exactly one run on input* $X_0, \ldots, X_{n-1}$ *of length* $\ell = \mathrm{LSUB}(\bigcup \mathbf{X})$.

Proof. Since A is runnable, it follows from **(3.**A**)** that A has at least one such run.

Suppose there are two runs $\mathbf{Y}$ and $\mathbf{Y}'$. Consider the set

$$K = \bigcup_{q \neq q' \in Q} [Y_q \cap Y'_{q'}],$$

The set of positions where $\mathbf{Y}$ and $\mathbf{Y}'$ differ. I assert that K is empty. Suppose not, then it must have a least element. This least element cannot be 0, since $|I| = 1$.

[5]Here we refer to systems with a notion of labelling and locality where we distinguish those labellings which can be augmented to ones that in every local region satisfy some constraint versus those that cannot. A wide variety of automata are of this form, including cellular automata, but a particularly enlightening example may be unranked tree automata, whose local regions are not bounded in size. Additionally, differential equations and operations are of this form: there is a sense in which they are a special case of continuous automata.

Then it is $S(x)$ for some x. Then we know the state of our runs at time x agree, but at time $S(x)$ disagree, contradicting δ being a function from $Q \times 2^n \to Q$. $\square$

Definition. *Given an automaton $A = (n, Q, I, \delta, F)$, define the* determinization *of A, denoted $\det(A)$, as the automaton:*

$$\det(A) = (n, \mathcal{P}(Q), \{I\}, \delta', \{K \subset n | K \cap F \neq \emptyset\}),$$

Where:

$$\delta'(R, \mathbf{s}, T) :\iff T = \{q' | \exists q \in R : \delta(q, \mathbf{s}, q')\}.$$

Of course $\det(A)$ is deterministic.

Proposition 24. *From our axioms, we can conclude that given an automaton $A = (n, m, I, \delta, F)$ and an input $X_0, \ldots, X_{n-1}$,*

$$\Phi_A(\mathbf{X}) \iff \Phi_{\det(A)}(\mathbf{X})$$

To summarize the proof: I will assert that at any point x, the state of $\det(A)$ will be the set of states for which there is a run of A on the same input with that state at point x. Showing that this follows from our axioms is a bit tricky: we can show that there is no specific point x where this stops happening, but our level of induction is not strong enough for this to suffice.

In order to show that states don't disappear in the run of $\det(A)$, that is, it isn't the case that there's a state which is reachable, but doesn't appear in the run of $\det(A)$, one simply has to compare a run containing the state in question at the position in question to the run of $\det(A)$. Bitwise operations will give us the set of positions where states of our run do not appear in the run of $\det(A)$ and this set cannot have a smallest element, so it's empty.

To show that states don't spuriously appear in the run of $\mathrm{det}(A)$, that is, it isn't the case that there's a state which isn't reachable, but does appear in the run of $\mathrm{det}(A)$ is trickier: given a state at a position in the run of $\mathrm{det}(A)$, we construct a run of A backwards, starting at this state, and using the run of $\mathrm{det}(A)$ as a guide to make sure that we stay within states that can be traced back to the starting state and not wind up in some dead-end branch. This construction is performed by an automaton H. We expect H to be reverse-runnable, but only because of our choice of input, so we need to add in a dummy state to ensure that H has a transition relation that makes it reverse-runnable, and then argue that for this particular input, the dummy state is never used.

So if there is an accepting run of A, it must be the case that the run of $\mathrm{det}(A)$ ends in a state containing the final state of our accepting run of A, hence this run of $\mathrm{det}(A)$ is accepting. If there is an accepting run of $\mathrm{det}(A)$ then it must be the case that some accept state appears in the final state of this run, and so there's a run of A such that this state appears as the final state.

It is also worth noting that this is the essential use of axiom schema **(3.**A**)** and **(4.**A**)**: asserting that the determinization has a run at all and then asserting that the auxiliary automaton H has a reverse run. Other uses of these axioms could be made into their own axioms with significantly less semantic content, but this case seems to use the full power of our schema.

Proof. $\mathrm{det}(A)$ is deterministic, so we may consider the unique run $\mathbf{Z}$ on input $\mathbf{X}$. I assert the following. Suppose $x \in Z_J$ for $J \in \mathcal{P}(m)$. Then J is exactly the set of all states q such that there is a run $Y_0, \ldots, Y_{m-1}$ satisfying $R_A(X_0, \ldots, X_{n-1}, x, Y_0, \ldots, Y_{m-1})$ with $x \in Y_q$. Informally, the state of $\mathrm{det}(A)$ at any one time x is the set of states for which there is some run of A with that state at time x.

It should be clear that if this statement is true for x then it's true for $S(x)$. Suppose it's the case that $x \in Z_J$ and J is exactly the set of states q that show up at position x in some run $\mathbf{Y}$ of A on input $\mathbf{X}$. Then by construction of δ', $S(x)$ is in some Z_K for K the set of states which potentially show up as the next states after reading the input at position x. One may construct an appropriate run by augmenting the runs $\mathbf{Y}$ by combining the singleton and boolean operation axioms.

Now suppose it's the case that there's a run $Y_0, \ldots, Y_{m-1}$ with state q appearing at position x but $x \in Z_J$ for some J with $q \notin J$. Then consider the set:

$$U := \{x | \exists i, J : x \in Y_i \wedge x \in Z_J \wedge i \notin J\} = \bigcup_{i \in m} \left[Y_i \setminus \bigcup_{J \ni i} Z_J \right].$$

This set exists by closure properties. By our assumption, this set is nonempty, and hence has a smallest element y. By construction of $\det(A)$, this $y \neq 0$. What this means is that there is an $S^{-1}(y)$ which is not in U but y is in U. But again, one can easily check that if $S^{-1}(y) \notin U$ then by construction of $\det(A)$, neither is y.

So the only thing that could go wrong was if there were a state q and a position x such that $x \in Z_J$ for some $J \ni q$, but there was no run $\mathbf{Y}$ of A with $x \in Y_q$. That is, some state mysteriously appeared in the run of $\det(A)$ that didn't belong there. Suppose this is the case. Consider the following automaton[6]:

$$H := (n + m, m \cup \{q_d\}, m \cup \{q_d\}, \delta'', \{q\}).$$

Where $(p, \mathbf{s}^\frown \mathbf{t}, p') \in \delta''$ if $(p, \mathbf{s}, p') \in \delta \wedge t_p = 1$ or if $p = q_d$ and there is no a such that $(a, \mathbf{s}, p') \in \delta$ and $t_a = 1$.

[6]This automaton is based on the *Harvester Automaton* for A used in *Generating the Regular Functions and Krohn-Rhodes*.

By construction, H is reverse-runnable, and we refer to q_d as the *dummy state*. Since H is reverse-runnable, there is a run $\mathbf{W}$ such that:

$$R_B(X_0, \ldots, X_{n-1}, U_0, \ldots, U_{m-1}, x, W_0, \ldots, W_m, W_{q_d}),$$

Where:

$$U_i := \{x | \exists J \ni i : x \in Z_J\} = \bigcup_{J \ni i} Z_J.$$

I assert that $W_{q_d} = \emptyset$. Suppose not, then W_{q_d} has a maximal element w. This element is not x, since we're in state q at that point.

Label our situation as follows: let J be the state of $\det(A)$ at position w and J' be the state of $\det(A)$ at position $S(w)$ (note that this even makes sense if $S(w) = x$). Let s be the input at time w. So, specifically, $w \in X_{\mathrm{s}}$, $w \in Z_J$, $w \in W_{q_d}$, $S(w) \in Z_{J'}$, and $S(w) \in W_r$. Here's a diagram to keep track of the situation. Recall that $\mathbf{X}$ was the original input to our automaton A, $\mathbf{Z}$ was the run of $\det(A)$ on input $\mathbf{X}$, and $\mathbf{W}$ was the run of H on input $(\mathbf{X}, \mathbf{U})$ where $\mathbf{U}$ was a representation of $\mathbf{Z}$.

	w	$S(w)$	
$\mathbf{X}$	s		
$\mathbf{Z}$	J	J'	
$\mathbf{W}$	q_d	r	(r')

Table 3.1: Transition Diagram for the Determinization and Harvester Near the Dummy State

On the one hand, because $w \in W_{q_d}$, it must be the case that there is no state a such that $(a, \mathrm{s}, r) \in \delta$ with $a \in J$.

- In case $S(w) = x$, r must be the final state of H which we chose to be q, which we know is in J'.

- Otherwise, r must have been obtained going backwards from another state, r', which by the transition relation for H tells us that $r \in J'$.

Either way, we get that $r \in J'$.

By construction of $\mathbf{Z}$, we know that any state showing up in J' can be traced back to a state in J. Specifically, there is some state a such that $(a, \mathrm{s}, r) \in \delta$ with $a \in J$, a contradiction.

This proves our assertion that W_{q_d} is empty, which means that $\mathbf{W}$ is a valid run of A on input $\mathbf{X}$ up to point x (part of the definition of H was that its output would have to satisfy the transition relation for A or have a q_d show up somewhere).

This completes the proof of our earlier assertion, that the state of $\det(A)$ at any one time x is the set of states for which there is some run of A up to that point with that state.

So, as stated earlier, if there is an accepting run of A, it must be the case that the run of $\det(A)$ ends in a state containing the final state of our accepting run of A, hence this run of $\det(A)$ is accepting. If there is an accepting run of $\det(A)$ then it must be the case that some accept state appears in the final state of this run, and so there's a run of A such that this state appears as the final state. $\square$

We will see in the next section that this means that it follows from our axioms that the standard complementation procedure of determinizing and then complementing the set of accept states works correctly.

3.7 Complementation

Definition. *Given a deterministic automaton* $A = (n, Q, I, \delta, F)$*, define* $\mathrm{Swap}(A) = (n, Q, I, \delta, Q \setminus F)$*.*

Proposition 25. *It follows from our axioms that if* A *is deterministic, and* $\mathbf{X}$ *a suitable input, then:*

$$\neg\Phi_A(\mathbf{X}) \iff \Phi_{\mathrm{Swap}(A)}(\mathbf{X}).$$

Proof. It follows from determinization that there is a unique run $\mathbf{Y}$ of A on input $\mathbf{X}$. One can determine the last state of this run by taking the maximum of $\bigcup \mathbf{Y}$, and determining where it came from. Either it came from F in which case $\Phi_A(\mathbf{X})$ or it came from $Q \setminus F$, in which case $\Phi_A(\mathbf{X})$, but not both. $\square$

Definition. *Given a general automaton* $A = (n, Q, I, \delta, F)$*, define* $\mathrm{Comp}(A) = \mathrm{Swap}(\det(A))$*.*

Proposition 26. *It follows from our axioms that for any* A *and suitable input* $\mathbf{X}$*:*

$$\neg\Phi_A(\mathbf{X}) \iff \Phi_{\mathrm{Comp}(A)}(\mathbf{X}).$$

3.8 Conjunction

In this section we show that if we have automata A and A' which correspond to formulas ϕ, ϕ', then we can construct an automaton $\mathrm{Conj}(A, A')$ which corresponds to the formula $\phi \wedge \phi'$.

Definition. *Given two* n*-ary automata* $A = (n, Q, I, \delta, F)$ *and* $A' = (n, Q', I', \delta', F')$ *define their* conjunction*:*

$$\mathrm{Conj}(A, A') := (n, Q \times Q', I \times I', \delta'', F \times F'),$$

Where:

$$\delta'' = \{((a,b), \mathbf{s}, (c,d)) | (a, \mathbf{s}, c) \in \delta \wedge (b, \mathbf{s}, c) \in \delta'\},$$

Proposition 27. *It follows from our axioms that, given n-ary A, A' and suitable $\mathbf{X}$, then:*

$$[\Phi_A(\mathbf{X}) \wedge \Phi_{A'}(\mathbf{X})] \iff \Phi_{\mathrm{Conj}(A,A')}(\mathbf{X}).$$

Proof. (⇒): Suppose we have runs $\mathbf{Y}$ witnessing $\Phi_A(\mathbf{X})$ and $\mathbf{Y}'$ witnessing $\Phi_{A'}(\mathbf{X})$. Construct run $\mathbf{W}$ where:

$$W_{(q,q')} = Y_q \cap Y'_{q'}.$$

It's easy to check that $\mathbf{W}$ witnesses $\Phi_{\mathrm{Conj}(A,A')}(\mathbf{X})$.

(⇐): Suppose we have run $\mathbf{Y}$ witnessing $\Phi_{\mathrm{Conj}(A,A')}(\mathbf{X})$. Then let:

$$W_q = \bigcup_{q' \in Q'} Y_{(q,q')},$$

And:

$$W'_{q'} = \bigcup_{q \in Q} Y_{(q,q')}.$$

Then it's easy to check that $\mathbf{W}$ witnesses $\Phi_A(\mathbf{X})$ and $\mathbf{W}'$ witnesses $\Phi_{A'}(\mathbf{X})$. □

3.9 Bounding and Unbounding

Just as important as the expressive equivalence between deterministic and non-deterministic automata is the expressive equivalence between bounded automata, those automata, like our implementation of logical automata, whose runs terminate immediately after reading the last nontrivial input, and unbounded automata, those automata which are allowed to continue running after reading the last nontrivial input. In this section, we will define an analog of Φ for running logical automata unboundedly, and provide constructions for an equivalence between bounded and unbounded automata.

Definition. *Recall that we constructed a formula*

$$R_A(X_0, \dots, X_{n-1}, \ell, Y_0, \dots, Y_{|Q|-1})$$

to indicate that $Y_0, \dots, Y_{|Q|-1}$ was a run of length ℓ of A on input $X_0, \dots, X_{n-1}$.

Define the formula $\Psi_{(n,Q,I,\delta,F)}(X_0, \dots, X_{n-1})$ in WS1S as:

$$\begin{aligned}\exists \ell : \exists Y_0, \dots, Y_{|Q|-1} : & \ell \geq \mathrm{LSUB}(\bigcup \mathbf{X}) \\ & \wedge R_{(n,Q,I,\delta,F)}(X_0, \dots, X_{n-1}, \ell, Y_0, \dots, Y_{|Q|-1}) \\ & \wedge \bigvee_{i \in F} \ell \in Y_i.\end{aligned}$$

We now define a notion of unbounding a logical automaton:

Definition. *Given an automaton $A = (n, Q, I, \delta, F)$, define its* unbounding*:*

$$\mathrm{UBd}(A) := (n, Q \times Q, \{(i,i) | i \in I\}, \delta', Q \times F),$$

Where:

$$\begin{aligned}\delta'((a,b), \mathbf{s}, (c,d)) :\iff & (\mathbf{s} = \mathbf{0} \wedge d = b \wedge \delta(a, \mathbf{s}, c)) \\ & \wedge (\mathbf{s} \neq \mathbf{0} \wedge d = c \wedge \delta(a, \mathbf{s}, c))\end{aligned}$$

This automaton keeps track of two things: the state of the original automaton A and the last state the automaton went into when it saw a nonempty input.

Proposition 28. *Given an automaton $A = (n, m, I, \delta, F)$, and suitable input $\mathbf{X}$, it is provable from our axioms that the following are equivalent:*

1. $\Phi_A(\mathbf{X})$,

2. $\Phi_{\mathrm{UBd}(A)}(\mathbf{X})$,

3. $\Psi_{\mathrm{UBd}(A)}(\mathbf{X})$.

Proof. (1) $\Rightarrow$ (2): Let $\mathbf{Y}$ be the accepting run as given in (1).

Given our accepting run $\mathbf{Y}$ of A, our goal is to construct two runs, which we will then combine together to produce an accepting run of $\mathrm{UBd}(A)$. This first is our run $\mathbf{Y}$. The second is the run above, but delayed in the case of inputs which are $\mathbf{0}$.

Define the automaton D:

$$D := (|Q| + 1, Q, I, \delta', Q),$$

Where:

$$\delta'(a, \mathbf{r}^\frown t, b) :\iff [(t = 1 \wedge r_b = 1) \vee (t = 0 \wedge b = a)].$$

Let $\mathbf{V}$ be the run of D on input $P''(Y_0), \ldots, P''(Y_{|Q|-1}), \bigcup_i X_i$.

One then checks that, by construction, $\mathbf{W}$ where:

$$W_{(q,q')} = Y_q \cap V_{q'},$$

Is an accepting run of $\mathrm{UBd}(A)$ on input $\mathbf{X}$.

(2) $\Rightarrow$ (3): This is trivial. A bounded run of $\mathrm{UBd}(A)$ is also an unbounded run of $\mathrm{UBd}(A)$.

(3) $\Rightarrow$ (1): Given an accepting unbounded run $\mathbf{Y}$ of $\mathrm{UBd}(A)$, let $\ell = \mathrm{LSUB}(\bigcup_i X_i)$. Then our desired run of A is $\mathbf{W}$, where:

$$W_q = \bigcup_{q'} Y_{(q,q')} \cap \mathrm{dcl}(\{\ell\}).$$

One can show inductively that everywhere beyond finishing reading the input $\mathbf{X}$, the second coordinate of the state $\mathrm{UBd}(A)$ is in is a particular accept state of A. One then argues that at position $\max(\bigcup_i X_i)$, $\mathrm{UBd}(A)$ read a nontrivial input, and so updated the second coordinate of its state to agree with the first coordinate of its state (ie, that particular accept state). If $\bigcup_i X_i$ is empty, then the second coordinate of the state cannot change, so we must have started in an accept state. Hence, the run $\mathbf{W}$ is accepting. □

We now define a notion of bounding a logical automaton:

Definition. *Given an automaton $A = (n, Q, I, \delta, F)$, define its* bounding*:*

$$\mathrm{Bd}(A) := (n, Q, I, \delta, F'),$$

Where:

$$\begin{aligned} F' = \{q \in Q | \exists i, q_0, \ldots, q_{i-1} :& q = q_0 \\ & \wedge q_{i-1} \in F \\ & \wedge \forall j : \delta(q_j, \mathbf{0}, q_{j+1}) \end{aligned}$$

Proposition 29. *It follows from our axioms that, given an automaton $A = (n, Q, I, \delta, F)$, and input $\mathbf{X}$, the following are equivalent:*

1. $\Psi_A(X)$,
2. $\Psi_{\mathrm{Bd}(A)}(X)$,
3. $\Phi_{\mathrm{Bd}(A)}(X)$,

Proof. (1) ⇒ (2): This is clear. Any witnessing run in (1) is also a witnessing run in (2).

(3) $\Rightarrow$ (1): One simply takes the witnessing run in (3) and augments it with the finite series of states which lead the final state of such a run to a final state of A. This augmented run is in our model by closure under finite modifications, and of course is an accepting unbounded run of A.

(2) $\Rightarrow$ (3): Consider an accepting run in (2). I asset that this run is all accept states from the end of reading the input. Specifically, suppose we have $\ell, Y_0, \ldots, Y_{|Q|-1}$ with $\ell \geq \mathrm{LSUB}(\bigcup X)$, $R_{\mathrm{Bd}(A)}(\mathbf{X}, \ell, \mathbf{Y})$, and $\ell \in Y_f$ for some $f \in F$. Define the set:

$$U := \bigcup_{i \notin F'} Y_i \setminus \mathrm{dcl}(\bigcup X),$$

Where F' is the set of accept states of $\mathrm{Bd}(A)$.

I assert that U, the set of all positions beyond reading the input where the state is not an accept state of $\mathrm{Bd}(A)$, is empty, and thus that $\mathrm{LSUB}(\bigcup X) \in Y_i$ for some $i \in F$, which would mean that the appropriate prefix of $\mathbf{Y}$ is the desired run in (3). Suppose U is nonempty. Then let u be the maximal element of U. u cannot be $\max(\bigcup Y)$, since we know that $\max(\bigcup Y) \in Y_f$ for some $f \in F'$ by hypothesis. So it must be the case that $S(u)$ is in Y_f for some $f \in F'$. In this case one may augment the sequence witnessing that $f \in F'$ to verify that whatever i gives us $u \in Y_i$ must also be in F', contradicting our choice of u. $\square$

In the next sections, we will take advantage of our new equivalence to handle a small detail regarding free variables and existential quantification (projection).

3.10 Permutation and Introduction of Variables

Our project so far has been to show that if we have a formula ϕ with free second order variables $X_0, \ldots, X_{n-1}$, then there is an automaton A with n inputs such that:

$$\forall X_0, \ldots, X_{n-1} : \phi(X_0, \ldots, X_{n-1}) \iff \Phi_A(X_0, \ldots, X_{n-1}).$$

However, this isn't quite all that we need to show. Consider the example of taking the conjunction of two formulas ϕ and ϕ', where ϕ has free variables X_0, X_1 and ϕ' has free variables X_1, X_2. We construct inductively, automata A, A' such that

$$\phi(X_0, X_1) \iff \Phi_A(X_0, X_1)$$

and

$$\phi'(X_1, X_2) \iff \Phi_A(X_1, X_2).$$

Before we can take our conjunctions, we must first construct automata B, B' that take in 3 inputs, such that

$$\phi(X_0, X_1) \iff \Phi_B(X_0, X_1, X_2)$$

and

$$\phi'(X_0, X_1, X_2) \iff \Phi_B(X_0, X_1, X_2).$$

Then we may use our main proposition from our section on conjunction to produce an automaton C such that

$$\phi(X_0, X_1) \wedge \phi'(X_1, X_2) \iff \Phi_C(X_0, X_1, X_2).$$

A small detail, certainly, but one we cannot brush over if we wish to prove that our equivalence holds from our axioms. There are many ways to handle this issue indirectly, but in this section we will address the issue directly with a pair of results.

Definition. *Given an automaton $A = (n, Q, I, \delta, F)$, define the automaton* $\mathrm{Aug}(A)$*:*

$$\mathrm{Aug}(A) := (n+1, Q, I, \delta', F),$$

Where:

$$\delta' := \{(a, \mathbf{s}^\frown t, b) | (a, \mathbf{s}, b) \in \delta\}.$$

Proposition 30. *Given an n-ary deterministic automaton A, for which an empty input takes accepting states to accepting states, the following are equivalent:*

1. $\Psi_A(X_0, \ldots, X_{n-1})$,
2. $\Psi_{\mathrm{Aug}(A)}(X_0, \ldots, X_{n-1}, X_n)$.

Proof. (2) $\Rightarrow$ (1) is immediate. The accepting run for (2) is the accepting run for (1).

(1) $\Rightarrow$ (2): The only difficult case here is if the accepting run for (1) is shorter than the new input X_n. Otherwise the accepting run for (1) is the accepting run for (2). Suppose now that we need to extend the accepting run for (1) to be longer, while still being accepting. We will prove that $\Phi_{\mathrm{Aug}(A)}(X_0, \ldots, X_n)$, from which (2) will follow. Since A is deterministic, so is $\mathrm{Aug}(A)$. By axiom **(3.A)**, there is a bounded run $\mathbf{Y}$ of $\mathrm{Aug}(A)$ on input $X_0, \ldots, X_n$. Since runs of deterministic automata are unique, $\mathbf{Y}$ is just an extension of the accepting run for (1). One then can prove by induction that since the accepting run for (1) ended in an accept state, and since the transition relation for $\mathrm{Aug}(A)$ on empty input takes accepting states to accepting states, $\mathbf{Y}$ must end in an accepting state. $\square$

Corollary 1. *Given an n-ary automaton A, the following are equivalent, for all $X_0, \ldots, X_n$:*

1. $\Phi_A(X_0, \ldots, X_{n-1})$,
2. $\Phi_{\det(A)}(X_0, \ldots, X_{n-1})$,
3. $\Psi_{\mathrm{UBd}(\det(A))}(X_0, \ldots, X_{n-1})$,
4. $\Psi_{\mathrm{Aug}(\mathrm{UBd}(\det(A)))}(X_0, \ldots, X_{n-1}, X_n)$,
5. $\Phi_{\mathrm{Bd}(\mathrm{Aug}(\mathrm{UBd}(\det(A))))}(X_0, \ldots, X_n)$.

Proof. All of the steps above are equivalences we've previously shown. It simply suffices for (3) $\iff$ (4) to note that any automaton in the image of UBd will take accept states to accept states, and that the unbounding of a deterministic automaton will be itself deterministic. $\square$

Finally, note that in our original motivating example, we had a formula $\phi'(X_1, X_2)$, and we wanted to, given an automaton A' such that:

$$\phi'(X_1, X_2) \iff \Phi_A(X_1, X_2),$$

construct an automata B' such that:

$$\phi'(X_1, X_2) \iff \Phi_B(X_0, X_1, X_2).$$

In order to obtain such an automaton in addition to our introduction of variables construction, we also need a permutation of variables construction. Simply note that the obvious construction of appropriately permuting the transition relations works correctly: one can show that the run of one automaton, by virtue of satisfying its transition relation, satisfies the transition relation of the other on suitably permuted inputs.

Having shown we can handle them, henceforth in this paper we will ignore issues of introduction and permutation of variables. In the next section, we apply our unbounding construction to prove that projected automata behave correctly, provably from our axioms.

3.11 Projection

In this section we seek to prove that the traditional construction for projecting the language accepted by an automaton works correctly in our axiom system. That is, we wish to show that given an automaton A, there is an automaton $\mathrm{Proj}(A)$ such that:

$$\exists X_n : \Phi_A(X_0, \ldots, X_{n-1}, X_n) \iff \Phi_{\mathrm{Proj}(A)}(X_0, \ldots, X_{n-1}).$$

Definition. *Given an automaton $A = (n+1, Q, I, \delta, F)$, define the unbounded projection:* $\mathrm{UProj}(A)$*:*

$$\mathrm{UProj}(A) = (n, Q \times \{0,1\}, I \times \{0,1\}, \delta', F \times \{0,1\}),$$

Where:

$$\delta' := \{((a,i), \mathbf{s}, (b,j)) | i, j \in \{0,1\}, (a, \mathbf{s}^\frown i, b) \in \delta\}.$$

Proposition 31. *Given suitable input $X_0, \ldots, X_{n-1}$, the following are equivalent:*

1. *$\exists X_n : \Psi_A(X_0, \ldots, X_n)$,*
2. *$\Psi_{\mathrm{UProj}(A)}(X_0, \ldots, X_{n-1})$.*

Proof. (1) $\Rightarrow$ (2): Suppose X_n is such that Φ_A has accepting run $\mathbf{Y}$ on input $X_0, \ldots, X_n$. Then construct run:

$$\begin{aligned} W_{(i,0)} =& Y_i \cap (\mathrm{dcl}(X_n \cup \bigcup_i Y_i) \setminus X_n), \\ W_{(i,1)} =& Y_i \cap X_n. \end{aligned}$$

I assert that this is an accepting run for $\Psi_{\mathrm{UProj}(A)}$. One can check that the transition relation holds.

(2) ⇒ (1): Suppose $\Psi_{\mathrm{UProj}(A)}$ has accepting run $\mathbf{Y}$ on input $X_0, \ldots, X_{n-1}$. Then construct:

$$\begin{aligned} W_i =& Y_{(i,0)} \cap Y_{(i,1)} \\ X_n =& \bigcup_i Y_{(i,1)} \end{aligned}$$

Then I assert that X_n is such that Ψ_A has accepting run $\mathbf{W}$ on input $X_0, \ldots, X_n$.

□

Finally, we construct the projection operation for bounded automata:

Proposition 32. *Given automaton A, the following are equivalent:*

1. $\exists X_n : \Phi_A(X_0, \ldots, X_n)$,
2. $\exists X_n : \Psi_{\mathrm{UBd}(A)}(X_0, \ldots, X_n)$,
3. $\Psi_{\mathrm{UProj}(\mathrm{UBd}(A))}(X_0, \ldots, X_{n-1})$,
4. $\Phi_{\mathrm{Bd}(\mathrm{UProj}(\mathrm{UBd}(A)))}(X_0, \ldots, X_{n-1})$.

As such, we refer to $\mathrm{Bd}(\mathrm{UProj}(\mathrm{UBd}(A)))$ as $\mathrm{Proj}(A)$.

This completes our collection of inductive constructions. We now need to provide, for various possible atomic formulae, base case automata that are provably equivalent to them.

3.12 Base Case Automata

While we've developed a lot of tools for constructing new automata from old, we haven't yet constructed many automata to start with.

Definition. *Define the deterministic automaton A_{Sub}:*

$$A_{\text{Sub}} := (2, \{s, f\}, \{s\}, \delta, \{s\}),$$

Where the b such that $\delta(a, \mathbf{s}, b)$ is given by the table below:

$\mathbf{s} \backslash a$	s	f
00	s	f
01	s	f
10	f	f
11	s	f

Table 3.2: Transition Table for A_{Sub}

Proposition 33. *It is provable from our axioms that $X \subseteq Y$ iff A_{Sub} accepts the pair X, Y.*

Proof. ($\Rightarrow$): Suppose that $X \subseteq Y$. Then one may prove that A_{Sub}, on reading X, Y, is always in the state s, since by the transition relation, there can be no $<$-least position when A_{Sub} switches from state s to state f. Hence it is accepting.

($\Leftarrow$): (by contrapositive) Suppose that $X \nsubseteq Y$. Then there is some $x \in X \setminus Y$. After this position, A_{Sub} will be in the f state, and by the transition relation there cannot be a $<$-least position when it returns to the s state, hence it will finish in the f state. □

Definition. *Define the deterministic automaton A_{Suc}:*

$$A_{\text{Suc}} := (2, \{s0, s1, f\}, \{s0\}, \delta, \{s0\}),$$

Where the b such that $\delta(a, \mathbf{s}, b)$ is given by the table below:

$\mathbf{s}\backslash a$	$s0$	$s1$	f
00	$s0$	f	f
01	$s1$	f	f
10	f	$s0$	f
11	f	$s1$	f

Table 3.3: Transition Table for A_{Suc}

Proposition 34. *It is provable from our axioms that $S''(X) = Y$ iff A_{Suc} accepts the pair X, Y.*

Proof. ($\Rightarrow$): Suppose that $S''(X) = Y$. Then I assert that an accepting run of A_{Suc} is $Z_{s0} = \text{dcl}(Y) \setminus Y$, $Z_{s1} = Y$ and $Z_f = \emptyset$. One can easily check that this satisfies the transition relation, and ends with an accept state.

($\Leftarrow$): Now suppose that $S''(X) \neq Y$, that is, there is some $x \in X$ such that $S(x) \notin Y$ or some $x \notin X$ such that $S(x) \in Y$. In either case, A_{Suc} will transition into state f after stage $S(x)$, and remain there until the end of the run. It can also be the case that there is no stage $S(x)$, that is, x is the largest element of $X \cup Y$, in which case our run will end on a $s1$ state, which is not accepting. □

3.13 The Weak Exclusively Second Order Theory of One Successor

We are now prepared to present automata A_ϕ for each formula ϕ with just second order free variables, such that

$$\phi(X_0, \dots, X_{n-1}) \iff \Phi_{A_\phi}(X_0, \dots, X_{n-1}).$$

This construction will take place in two steps: first we will reduce ϕ to include only second order variables, and those logical atoms and connectives that we know how to deal with ($\subseteq, S'', \wedge, \neg, \exists$) and then translate, inductively, to the automaton A_ϕ.

Definition. *Define the* Weak Exclusively Second Order Theory of One Successor *(W!S1S) as WS1S, augmented with the additional relations: $S'(X,Y)$ which indicates that $S''(X) = Y$, and $\subseteq$, without:*

- *First order variables and quantification,*
- *$\in$ and S,*
- *Disjunction,*
- *Universal quantification.*

Proposition 35. *Any formula ϕ in WS1S with only second order free variables has an equivalent in W!S1S.*

Proof. Our first step will be to replace first order variables with second order ones, representing the singletons of their first order parts. Let $\text{Sing}(X)$ be shorthand for:

$$\forall Y : Y \subseteq X \implies [(X \subseteq Y) \vee (\forall Z : Y \subseteq Z)].$$

- Replace first order variables x with X,
- Replace atomic formulas of the form $x \in Y$ with $X \subseteq Y$,
- Replace terms of the form $S(x)$ with $S''(X)$,
- For nested uses of S'', introduce new variables to represent intermediate terms. e.g.:

$$Y = S''(S''(X)) \rightsquigarrow \exists Z : Y = S''(Z) \wedge Z = S''(X),$$

- Relationalize S'', replacing instances of the form $S''(X) = Y$ with $S'(X, Y)$,
- Replace equality with $\subseteq$:

$$X = Y \rightsquigarrow X \subseteq Y \wedge Y \subseteq X,$$

- Replace universal quantifiers with conjugated existential quantifiers,
- Replace first order existential quantifiers:

$$\exists x : \phi \rightsquigarrow \exists X : \mathrm{Sing}(X) \wedge \phi,$$

- Replace disjunctions with conjugated conjunctions:

$$\phi \vee \psi \rightsquigarrow \neg((\neg\phi) \wedge (\neg\psi)).$$

The resulting formula will be equivalent to the original. Note that the lack of spurious singletons is critical to the success of this equivalence. □

Definition. *Given a formula ϕ in W!S1S, define the automaton A_ϕ inductively:*

- *If $\phi := S'(X, Y)$, then $A_\phi = A_{\mathrm{Suc}}$.*
- *If $\phi := X \subseteq Y$, then $A_\phi = A_{\mathrm{Sub}}$.*
- *If $\phi := \psi \wedge \tau$, then augment and permute A_ψ and A_τ appropriately so their free variables match, and take the conjunction of the resulting automata.*
- *If $\phi := \neg\psi$, then let $A_\phi = \mathrm{Comp}(A_\psi)$.*
- *If $\phi := \exists X : \psi$ then, modulo suitable variable permutations, let $A_\phi = \mathrm{Proj}(A_\psi)$.*

Combining all of our theorems from before, we can prove inductively that:

$$\phi(X_0, \ldots, X_{n-1}) \iff \Phi_{A_\phi}(X_0, \ldots, X_{n-1}).$$

Definition. *For formulas ϕ of WS1S with only free second order variables, we produce A_ϕ by first producing ϕ' in W!S1S which is equivalent, then computing $A_{\phi'}$.*

In the next section, we will finally prove that our axiomatization is complete:

3.14 0-ary Automata

Up to this point, we have provided a canonical form for formulas of WS1S with only free second order variables (namely, for a formula ϕ its canonical form is Φ_{A_ϕ}), and shown that it follows from our axiom system that every such formula is equivalent to its canonical form. Now, in order to prove that our axiom system is complete, we must show that if a sentence ϕ is true (in the theory WS1S), then its canonical form is provable.

First, we must examine what the canonical form of a sentence can look like. Specifically, it looks like a 0-ary automaton, an automaton that takes in no input. What's more, because none of its inputs contain elements, the length ℓ of any run of this 0-ary automaton is $\mathrm{LSUB}(\emptyset) = 0$. So a run starts and ends in the same state. If we unpack the definitions, we find that our 0-ary automaton is accepting iff it has a start state which is also an accept state.

Proposition 36. *Given 0-ary automaton $A = (0, Q, I, \delta, F)$,*

$$\Phi_A \iff I \cap F \neq \emptyset.$$

What's more, this follows from our axiomatization.

Proof. ($\Rightarrow$): Given an accepting run $\mathbf{Y}$ of A, we note that by definition, $\mathbf{Y}$ is a partition of $\{0\}$. Let $q \in Q$ be such that $0 \in Y_q$. Then by construction, $q \in I \cap F$.

($\Leftarrow$): Given a q in $I \cap F$, then I claim that $\mathbf{Y}$ where $Y_q = \{0\}$ and all other $Y_{q'}$ are empty is an accepting run. $\square$

As such, any sentence in canonical form which is true is provable. And thus, any sentence which is true is provably equivalent to a sentence in canonical form, which is provable. Hence any sentence in WS1S which is true is provable from our axiom system. One can also verify the soundness of our axiom system. We summarize this in the following theorem:

Theorem 7. *Our proposed axiom system is sound and complete for WS1S.*

3.15 Finitely Many Axioms Do Not Suffice

Our axiom system is fairly small and simple. A significant improvement, however, would be to be able to produce a finite axiomatization of WS1S. In this section, we prove that no such axiomatization is possible, by providing a structure H_p which satisfies all of our axioms up to a point, but does not satisfy all of our axioms.

Definition. *Let $\omega + \zeta$ refer to the collection of elements of the form $n \in \omega$ or $\infty + z | z \in \mathbb{Z}$, with natural ordering such that every element of the form $\infty + z$ (elements in the ζ-*part*) is $>$ every element of the form $n \in \omega$ (elements in the ω-*part*). We also introduce the natural S operation which takes elements of the form n to $n+1$ and elements of the form $\infty + z$ to $\infty + z + 1$.*

Definition. *A subset X of $\omega + \zeta$ is said to be* internally periodic *of period m if there is some $a \in \omega$ and $b \in \mathbb{Z}$ and some set $P \subseteq \{0, \ldots, m-1\}$, such that an element of*

$\omega + \zeta$ between a and $\infty + b$ is in X iff it is of the form c or $\infty + c$ for some c congruent to an element of P modulo m.

Definition. *Let p be an odd prime. Let W_p denote the collection of bounded subsets of $\omega + \zeta$, which are internally periodic of some period not divisible by p. Then let H_p denote the two-typed structure:*

$$H_p := (\omega + \zeta, W_p, \in, S).$$

Lemma 9. *In H_p, the downward closed sets are exactly those sets of the form $\{x | x < \ell\}$ for some $\ell \in \omega + \zeta$.*

Proof. It is clear that sets of this form are downward closed. It suffices to show that if a second order object of H_p is downward closed, then it is of this form. Suppose the set $X \in W_p$ is downward closed. Then it must be internally periodic of period 1. Since it's in W_p it must also be bounded above by some $x \in \omega + \zeta$. Suppose X contains any element of the ζ part. Then by properties of $\mathbb{Z}$, we can find a largest element in its ζ part, and thus a largest element $w \in X$. By downward closure this set must contain all elements of the ζ part up to w. By internal periodicity, X must contain a final segment of elements of the ω part. And by downward closure again, X must contain all elements of the ω part. Thus X is of the desired form.

Suppose instead that X does not contain any element of the ζ part. By internal periodicity, it must thus be bounded above in the ω part. One can then reason that any downwards closed subset of ω which is bounded above must be of the desired form. □

Proposition 37. *H_p is a model of axioms* **(1.1)** *through* **(2.5)**

Proof. Recall that our axioms **(1.1)** through **(1.7)** classified those linear orderings which were elementarily equivalent to $(\omega, <)$. As mentioned in [9], these are the linear orderings of the form $(\omega + \zeta \cdot L)$, in particular, this includes $\omega + \zeta$, the structure we're working in now. It suffices to prove that the construction of $<$ in terms of downward closed sets gives us the same $<$ as the natural ordering on $\omega + \zeta$, which follows immediately from the above lemma.

One then routinely checks axioms **(2.1)** through **(2.5)**:

(2.1): Second order elements in H_p are sets and thus satisfy extensionality.

(2.2): Singletons are bounded and internally periodic of period 1.

(2.3): The union of two bounded internally periodic sets of periods a, b is bounded and internally periodic of period ab. If $p \nmid a, b$ then $p \nmid ab$.

(2.4): The difference of two bounded internally periodic sets of periods a, b is bounded and internally periodic of period ab.

(2.5): By definition, all second order elements in H_p are bounded above. □

Proposition 38. *H_p is a model of* **(3.**A**)** *for any A with fewer than p states.*

Proof. Given a runnable automaton A, first, restrict to an automaton A' which is runnable and deterministic by removing terms from its transition relation. Any accepting run of A' on suitable input $X_0, \ldots, X_{n-1}$ will also be a run of A. Let $X_0, \ldots, X_{n-1}$ be from W_p, and let us construct run $Y_0, \ldots, Y_{|Q|-1}$. If none of the $X_0, \ldots, X_{n-1}$ actually intersect the ζ part, just use the theorem for the standard model of WS1S, so we may assume at least one of the X_i has an element from the ζ part. First let us construct the run restricted to the ω part. By runnability and induction on ω, we can construct the unique run restricted to the ω part. I claim that this run is eventually periodic of a period not divisible by p. Since each of the inputs $X_0, \ldots, X_{n-1}$ restricted to their ω parts are eventually periodic of a period not divisible by p, then beyond some point $x \in \omega$, all of the $X_0, \ldots, X_{n-1}$ will together be periodic of a period m which is a factor of the product of the periods of each of the X_i, hence also not divisible by p. Since A' is deterministic, its run on this input will eventually be periodic of period k times m for some k less than or equal to the number of states of A, which is less than p. Hence, this run will have period not divisible by p.

Now we need to continue the run of A into the ζ part. The X_i will together be initially periodic of period m up to some point $\infty + z$ where one of them first diverges from its periodic behavior. Continue the Y_i up to this point by throwing $\infty + w$ into Y_i iff elements of the periodic portion of the ω part of the Y_i are congruent to w modulo the collective period of the Y_i. I claim that this still satisfies the transition relation, because the identical periods in the ω part satisfied the transition relation. After the point $\infty + z$, continue the run inductively just following the transition function up until the end of the input. □

Proposition 39. *H_p is a model of* **(4.**A**)** *for any A with fewer than p states.*

The proof is similar to the above, but in reverse.

Proposition 40. *H_p is not a model of WS1S.*

Proof. One can write a formula $\tau_p(X)$ which says:

$$\tau_p(X) := 0 \in X \wedge \bigwedge_{i=1}^{p-1} S^i(0) \notin X$$
$$\wedge \forall x : S^p(x) \in X \implies x \in X.$$

That is, X contains 0 and exactly every pth element, up to some point. It is a fact of WS1S that

$$\forall x : \exists y > x : \exists X : \tau_p(X) \wedge y \in X.$$

That is, for any x, we can always find such a periodic set containing an element larger than x. However, this is clearly not true of H_p for x in the ζ part. □

As such, we can find models of arbitrary finite subsets of our axiom system which are not models of the whole thing.

Theorem 8. *WS1S is not finitely axiomatizable.*

Proof. Suppose we have a finite axiomatization of WS1S. Take its conjunction to produce a single axiom ϕ for WS1S. Since our axiom system is complete, ϕ must follow from some finite subset of it. For some large enough p, H_p will satisfy this finite set of axioms. So ϕ is true of H_p. This contradicts our assumption that ϕ was an axiomatization for WS1S, since it is true of a structure which is not a model of WS1S. □

3.16 Conclusion and Future Research

As we can see from this example, this axiom system is well suited for proving that H_p is nearly a model of WS1S. Compare with an axiom system similar to that in [10], which would require proving induction for every WS1S formula. In chapter 4, we will use this axiom system to investigate what nonstandard models WS1S has.

It should be clear from our discussion in section 2 that we could just as easily be studying the weak second order theory of $(\omega, <)$. One may then ask about the weak second order theories, or just the second order theories in general, of other linear orderings. Is there, for instance, a notion of automaton that takes inputs which instead of being labeled finite segments of ω, were labeled intervals of $\mathbb{Q}$ as a linear ordering? Do such automata provide a canonical form for weak second order formulas of $(\mathbb{Q}, <)$? Perhaps such automata, if they exist, would be related to continuous automata.

One may also ask whether this axiomatization is minimal. An effort has been made to avoid redundancy among axioms **(1.1)** through **(2.5)**, although whether there is some residual redundancy and how many of these axioms follow from axioms **(3.**A**)** or **(4.**A**)** are still matters for study. A proof of minimality would require producing a large number of structures which satisfy various subsets of our axiomatization.

Finally, we saw some examples of relations on first order elements of WS1S that essentially required the use of second order quantification in their definitions. Specifically, $<$ which played a critical role in our axiomatization, and modular congruence modulo a fixed natural number, which played a critical role in the construction of our nearly-models H_p. One may then ask if there are any more which cannot be expressed in terms of these two.

Although WS1S is not nearly as expressive as larger axiomatized theories such as Peano Arithmetic or ZFC, it is easily decidable and its nonstandard models are tractable (yet still, as we will see in the next chapter, interesting).

CHAPTER 4

REGARDING NONSTANDARD MODELS OF THE WEAK SECOND ORDER THEORY OF ONE SUCCESSOR

In this chapter, we answer two key questions about nonstandard models of the weak second order theory of one successor (WS1S): What can their first-order parts look like, and what can their second order parts look like for a particularly interesting case (when the first order part is $\omega + \zeta$). We will also provide some tools for constructing nonstandard models of WS1S from already known nonstandard models. This will go a long way towards establishing a complete classification of the nonstandard models of WS1S.

4.1 Introduction

Recall that the *weak second order theory of one successor*, WS1S, is the theory of the two-typed structure $G := (\omega, \mathcal{F}(\omega), \in, S)$, where $\mathcal{F}(\omega)$ is the set of finite subsets of ω, $\in$ is containment, and S is the successor operation on ω. As usual, lowercase variables will denote elements of the first type, uppercase variables will represent elements of the second type, and boldface variables will indicate shorthand for finite or infinite sequences of elements of the corresponding type indexed by natural numbers. Crucially, these will not be indexed by first order elements from the current model of WS1S, but standard natural numbers. This theory is of interest because it is a decidable second order theory of ω, and equally expressive to finite automata.

Recall that in the previous chapter, we proved that WS1S was not finitely axiomatizable, and provided an axiomatization with schema indexed by finite automata. This axiomatization is well suited to verifying that various structures satisfy it and thus are nonstandard models of WS1S.

A nonstandard model of WS1S is simply a structure which satisfies all of the same sentences as the standard (or *ground*) model G as defined above, but which is not the standard model. Nonstandard models are a useful tool for understanding the limits of the expressiveness of a language. Our nonstandard models will clearly differ from our ground model G, but not in the WS1S formulas they satisfy. That is, the ways that our nonstandard models differ from G or from each other will not be expressible in the language of WS1S.

We expect to find a large number of nonstandard models of WS1S. The Löwenheim-Skolem Theorem says that if a countable first-order theory has an infinite model, then it has one of every infinite cardinality. Since we can view WS1S as the first-order theory of the two-typed structure G, this gives us nonstandard models of WS1S of every infinite cardinality.

A priori, studying nonstandard models of second order theories seems like a hefty task: in a nonstandard model H, even if one has a grasp on the collection of first order objects, the second order objects are some nebulous collection related to the first order objects via H's interpretation of $\in$, $\in_H$. Fortunately, nonstandard models of second order theories satisfy extensionality, so the second order elements are determined by which first order elements they $\in_H$-contain. This allows us to identify second order elements with the collection of their $\in_H$-elements. So up to isomorphism, a nonstandard model of WS1S will be of the form $(M_1, M_2, \in, S)$ where M_2 is a collection of subsets of M_1, $\in$ is genuine containment, and S is some interpretation of S on M_1. Combined with our understanding of nonstandard models of $(\omega, <)$, this will give us a very constructive representative for every isomorphism class of nonstandard model of WS1S.

Once we have this constructive form for representatives of isomorphism classes of nonstandard models of WS1S, we need to decide which structures of this constructive form actually are nonstandard models of WS1S. We will prove that for every nonstandard model L of $(\omega, <)$, there is a nonstandard model of WS1S whose first order elements, ordered by that model's interpretation of $<$ form the linear ordering L. Then we will examine what second order parts are available in the simplest nonstandard case: where the first order part is the linear ordering $\omega + \zeta$.

Finally, we will provide tools for cutting apart nonstandard models of WS1S and gluing them back together to produce other nonstandard models.

4.2 Recalling our Axiomatization

Recall that our axiomatization from *Axiomatizing the Weak Second Order Theory of One Successor* made heavy use of shorthand. In this section, we reintroduce this shorthand, and present a compact list of our axioms.

Definition. *A set X is said to be* downward closed *if:*

$$\mathrm{dc}(A) :\equiv \forall X : S(x) \in A \implies x \in A$$

We define a linear ordering on the first order elements by:

$$x \leq y :\equiv \forall X : \mathrm{dc}(X) \wedge y \in X \implies x \in X$$

Definition. *Given two sets A, B, we write:*

$$C = A \cup B :\equiv \forall x : x \in C \iff x \in A \vee x \in B$$

$$C = A \cap B :\equiv \forall x : x \in C \iff x \in A \wedge x \in B$$

$$C = A \setminus B :\equiv \forall x : x \in C \iff x \in A \wedge x \notin B$$

$$A \subseteq B :\equiv \forall x : x \in A \implies x \in B$$

Recall that we had a notion of automaton:

Definition. *A* logical automaton *is a 5-tuple $A = (n, Q, I, \delta, F)$ where:*

- *n is the number of inputs, potentially 0.*
- *Q is a finite set of states. We may occasionally refer to the elements of Q via some canonical ordering as $0, \ldots, |Q| - 1$, but more often we will let the elements of Q retain some additional structure. We will also refer to the number of states of an automaton A as $|A|$. Often we will prove results for automata with state set m where this suffices to prove the result in general.*
- *$I \subseteq Q$ is the set of* initial states.
- *$\delta \subseteq Q \times 2^n \times Q$ is the* transition relation.
- *$F \subseteq Q$ is the set of* final states.

Definition. *Define the formula $R_{(n,Q,I,\delta,F)}(X_0, \ldots, X_{n-1}, \ell, Y_0, \ldots, Y_{|Q|-1})$ in WS1S as the conjunction of:*

- $\forall x : \bigvee_{i \in Q} \bigwedge_{j \in Q} [x \in Y_j \iff (i = j \wedge 0 \leq x \leq \ell)]$
- $\bigvee_{i \in I} 0 \in Y_i$
- $\forall x : 0 \leq x < \ell \implies \left[\bigvee_{(i,\mathbf{s},k) \in \delta} (x \in Y_i \wedge x \in X_{\mathbf{s}} \wedge S(x) \in Y_k)\right]$

This formalizes the notion of $(\ell, Y_0, \ldots, Y_{|Q|-1})$ being a run of (n, Q, I, δ, F) on input $X_0, \ldots, X_{n-1}$.

Definition. *Define the formula* $\Phi_{(n,Q,I,\delta,F)}(X_0, \ldots, X_{n-1})$ *in WS1S as:*

$$\begin{aligned} \exists \ell : \exists Y_0, \ldots, Y_{|Q|-1} : & \ell = \mathrm{LSUB}(\bigcup \mathbf{X}) \\ & \wedge R_{(n,Q,I,\delta,F)}(X_0, \ldots, X_{n-1}, \ell, Y_0, \ldots, Y_{|Q|-1}) \\ & \wedge \bigvee_{i \in F} \ell \in Y_i. \end{aligned}$$

Where $\mathrm{LSUB}(X)$ *is shorthand for the least strict upper bound for a set, either* 0 *for the emptyset or* $S(\max(X))$ *for a nonempty set.*

This formalizes the notion of the automaton (n, Q, I, δ, F) accepting inputs $X_0, \ldots, X_{n-1}$. Of course, not every automaton will have an accepting run on every input, but we can expect certain classes of automata to at least have runs on every input:

Definition. *An automaton* (n, Q, I, δ, F) *is said to be* runnable *if:*

- $|I| > 0$,
- $\forall q, \mathrm{s} \in 2^n : \exists q' : \delta(q, \mathrm{s}, q')$.

Definition. *An automaton* (n, Q, I, δ, F) *is said to be* reverse-runnable *if:*

- $|F| > 0$,
- $\forall q, \mathrm{s} \in 2^n : \exists q' : \delta(q', \mathrm{s}, q)$.

Finally, we can present our axiom system:

(1.1) $\forall x, y, z : x < y \land x < z \implies x < z$ (< is transitive)

(1.2) $\forall x, y : x < y \implies \neg(y < x)$ (< is antisymmetric)

(1.3) $\forall x, y : x < y \lor x = y \lor y < x$ (< is total)

(1.4) $\forall x : S(x) > x$ (S is strictly increasing)

(1.5) $\forall x : \nexists y : x < y < S(x)$ (S is a successor)

(1.6) $\exists o : \forall x : o \leq x$ (< has minimal element)

(1.7) $\exists o : \forall x : x \neq o \implies \exists y : S(y) = x$ (S has image $\omega \setminus \{0\}$)

(2.1) $\forall X, Y : (\forall x : x \in X \Leftrightarrow x \in Y) \Rightarrow X = Y$ (Extensionality)

(2.2) $\forall x : \exists X : \forall y : y \in X \Leftrightarrow y = x$ (Singleton)

(2.3) $\forall X, Y : \exists Z : \forall x : x \in Z \Leftrightarrow (x \in X \lor x \in Y)$ (Union Closure)

(2.4) $\forall X, Y : \exists Z : \forall x : x \in Z \Leftrightarrow (x \in X \land x \notin Y)$ (Difference Closure)

(2.5) $\forall X \neq \emptyset : \exists x : x \in X \land \forall y : y \in X \Rightarrow y \leq x$ (Maximal Element)

(3.A) $\forall X_0, \ldots, X_{n-1} : \Phi_{(n,Q,I,\delta,Q)}(X_0, \ldots, X_{n-1})$ (A Runnable)

(4.A) $\forall X_0, \ldots, X_{n-1} : \Phi_{(n,Q,Q,\delta,F)}(X_0, \ldots, X_{n-1})$ (A Reverse-Runnable)

4.3 L-Models

In this section, we introduce the notion of an L-model, a very constructive form of nonstandard model of WS1S that we will use as representatives for isomorphism classes of our nonstandard models. Recall that nonstandard models of $(\omega, <)$ are of the form $\omega + \zeta \cdot L$ [9]. An L-model will be one whose first order part is $\omega + \zeta \cdot L$.

Definition. *A two-typed structure* $(M_1, M_2, \in, S)$ *is said to be an* L-model *if:*

$$M_1 = \omega + \zeta \cdot L$$

Where $\zeta = (\mathbb{Z}, <)$*, the linear ordering of the integers, L is a linear ordering, and* $\cdot$ *is the* *lexicographic product, so:*

$$\omega + \zeta \cdot L = \omega + \underbrace{\zeta + \cdots + \zeta}_{L},$$

Specifically, M_1 contains elements of the form $n \in \omega$, and elements of the form (a, b) with $a \in \mathbb{Z}$ and $b \in L$, with elements from ω ordered naturally, being less than elements of the form (a, b). Elements of the form (a, b) are ordered first by second coordinate and then by first coordinate. And:

1. *$M_2 \subseteq \mathcal{P}(M_1)$, and $\in$ is ordinary containment between M_1 and M_2,*
2. *S is ordinary successor on ω and takes $(a, b) \in \zeta \times L$ to $(a + 1, b)$,*
3. *Every set of the form $\{x | x < \ell\}$ is in M_2.*
4. *The only sets X in M_2 which, when intersected with each ω or ζ part form an initial (possibly empty) interval of that part are of the form $\{x | x < \ell\}$.*

Proposition 41. *In every L-model, $<$ as defined on M_1 agrees with $<$ as defined in terms of downwards (predecessor) closed elements.*

Proof. Suppose that $x \leq y \in \omega + \zeta \cdot L$, and we have a predecessor closed set X containing y. Since X is predecessor closed, when intersected with any ω or ζ part, it will form a (possibly empty) initial interval. Hence X is of the form $\{w | w < \ell\}$. By transitivity of $<$, x must also be in this set.

Suppose now that every predecessor closed set X containing y contains x. Since our model contains $\{w | w \leq x\}$ and this is a predecessor closed set containing x it must also contain y. Hence $y \leq x$. □

If a structure is an L-model, that represents a significant start towards proving that it satisfies all of our axioms.

Proposition 42. *Every L-model satisfies axioms* **(1.1)** *through* **(1.7)**, *and* **(2.1)**.

Proof. Our condition that $<$ as defined in terms of downward (predecessor) closed sets agrees with $\leq$ as the linear ordering on $\omega + \zeta \cdot L$ will go a long way. By virtue of being a linear ordering $<$ will satisfy axioms **(1.1)**, **(1.2)**, and **(1.3)**.

One checks that when we interpret $<$ as the linear ordering on $\omega + \zeta \cdot L$ and S as the natural successor operation on $\omega + \zeta \cdot L$, axioms **(1.4)** through **(1.7)** are satisfied. Thus, when we interpret $<$ as the equivalent linear ordering defined in terms of predecessor closed collections, and S as the natural successor operation on $\omega + \zeta \cdot L$, axioms **(1.4)** through **(1.7)** are satisfied.

Finally, **(2.1)** is satisfied by virtue of condition (1) and the fact that sets satisfy extensionality. □

Proposition 43. *G is the unique 0-model of WS1S. (We use natural numbers n for the linear order of size n whose elements are $0, \ldots, n-1$.)*

Proof. G clearly is a 0-model (one whose first order type is ω) which models WS1S. Suppose there is another 0-model G' which is a model of WS1S. Any second order element of G will be a finite set of natural numbers, so one may construct a formula of WS1S explicitly stating that that second order element is an element of our model. As such the second order elements of G' must contain as a subset the second order elements of G. Furthermore, G' cannot contain any second order elements which are infinite subsets of ω, since these aren't bounded, violating our maximal element axiom. As such, $G' = G$. □

Proposition 44. *Every model of WS1S is isomorphic to an L-model for some L.*

Proof. Given a model $(M_1, M_2, \in, S)$, define an equivalence relation $\sim$ on M_1 where:

$$x \sim y :\equiv \exists n, m : S^n(x) = S^m(y).$$

Note that $\leq$, as defined in terms of downward closed (ie, predecessor closed) elements of M_2, respects these equivalence classes. That is, if $x \leq y$ and $s' \sim x$ and $y' \sim y$, then $x' \leq y'$ or $x' \sim y'$. Pick a representative of each equivalence class, including 0 (the unique element not in the image of S) for its equivalence class, and denote the set of representatives of other equivalence classes by L. L admits a natural linear ordering inherited by $\leq$ on M_1, as per above. Every element of M_1 is thus $S^n(0)$ or $S^n(\ell)$ or $S^{-n}(\ell)$ for some $\ell \in L$, which we will map into in our isomorphism $n \in \omega$ or (n, ℓ) or $(-n, \ell)$ respectively. This clearly respects S.

Condition (3) holds because these sets are definable. Condition (4) is equivalent to saying that all predecessor closed sets are of the form $\{x | x < \ell\}$, which we can write in the language of WS1S and which is true of our ground model G, so is in the theory of WS1S.

We will identify any element of M_2 with the set of its $\in$-elements. This association is injective by extensionality. It is not surjective. This clearly respects $\in$. □

Of course, this isn't an assurance that there are L-models for every L. In the case of nonstandard models of the first order theories of $(\omega, +)$ and $(\omega, +, \cdot)$, we can still define $<$, and there are some very strict restrictions on what sorts of linear order types $<$ can have. Specifically, nonstandard models of $(\omega, +)$ must have linear order type $\omega + \zeta \cdot L$ for some dense L. We will address this restriction in the next section.

4.4 Minimal L-Models

Recall that we have a notion of modular congruence in WS1S. Given a natural number n (not an element of our model), we can construct the formula:

Definition. *Given an L-model $(M_1, M_2, S, \in)$ of WS1S, we write*

$$\mu_m(X, x) :\equiv \forall y : S^m(y) \leq x \implies [S^m(y) \in X \iff y \in X].$$

And, for $n < m$,

$$M_{n,m}(x) :\equiv \forall Y : x \in Y \wedge \mu_m(Y, x) \implies S^n(0) \in Y.$$

Proposition 45. *The following are part of the theory of WS1S:*

1. *For all $M_{n,m}(x) \iff M_{(n+1 \mod m),m}(S(x))$.*
2. *For all $m \in \omega, x \in \omega + \zeta \cdot L$, exactly one of $M_{i,m}(x)$ is true for $i = 0, \ldots, m-1$.*
3. *For all m, n, exactly one of $M_{n,m}(S^i(x))$ for $i = 0, \ldots, m-1$.*
4. *For all m, n, $M_{n,m}(x) \iff M_{n,m}(S^m(x))$.*

In agreement with G, we say that a first order element x in a specific model is *congruent to n modulo m* if $M_{n,m}(x)$. Also, note that in an L-model, specifying the congruences of the elements of the form $(\ell, 0)$ is enough to determine the congruences of all elements.

What's more, once we've specified L and these congruences, we have enough information to throw in just those second order elements which are absolutely necessary to construct an L-model. But first, we need a language for specifying our congruences:

Definition. *A sequence $\{s_n\}_{n>2\in\omega}$ is said to be a* congruence type *if:*

- $s_n < n$
- *if $a|b$ then $s_b \equiv s_a \mod a$.*

We will denote the set of all congruence types $\mathbb{K}$.

Note that the sequence $\mathbf{0} := \{0\}_{n>2}$ is a congruence type, although there are continuum many other such sequences[1].

Once we have a particular association of congruence types to all of our elements in mind, we can define modular equivalence classes for every element of $\omega + \zeta \cdot L$:

Definition. *Given L, $k : L \to \mathbb{K}$, define $M^k_{n,m}$:*

$$M^k_{n,m}(x) :\equiv \begin{cases} (r \equiv n \mod m) & x = r \in \omega \\ (k(\ell)_m + r \equiv n \mod m) & x = (r, \ell) \end{cases}$$

If $M^k_{n,m}(x)$, we say that x is congruent to n modulo m.

Alternately, for clarity and when k is given, we write $x \bmod m = n$, with the note that m and n are natural numbers, and not general elements of our models.

Note that:

$$M^k_{k(\ell)_m, m}((0, \ell)),$$

That is, $(0, \ell)$ is congruent to $k(\ell)_m$ modulo m.

[1]One may choose remainders modulo primes arbitrarily, and the congruence type is determined completely by remainders modulo powers of primes (via the Chinese Remainder Theorem), which must themselves be interconsistent. Remainders modulo p^a determine remainders modulo p^b for $b < a$.

Definition. *Given a linear ordering L, a map $k : L \to \mathbb{K}$, a tuple $x_0, \ldots, x_{n-1}$ of elements of $\omega + \zeta \cdot L$, a tuple $m_0, \ldots, m_{n-2}$ of natural numbers > 1, and a tuple $p_0, \ldots, p_{n-2}$ of subsets where $p_i \subseteq \{0, \ldots, m_i - 1\}$, define:*

$$\mathrm{Pw}(k, \mathbf{x}, \mathbf{m}, \mathbf{p}) := \bigcup_i : \{y \in [x_i, x_{i+1}) | y \bmod m_i \in p_i\}.$$

This is a second order element whose behavior is piecewise periodic. Each piece is an interval of the form $[x_i, x_{i+1})$, the m_i and p_i specify the periodic behavior on each of those intervals, and the map k describes how infinite periodic behavior which is terminal in one ω or ζ component gets picked up in the next. We will construct our model consisting entirely of second order elements of this form for arbitrary $\mathbf{x}, \mathbf{m}$, and $\mathbf{p}$, but for fixed k. It's necessary that we have a clear sense of the congruence class of each of our infinite elements: in order to be a model of WS1S, it is necessary that we are not lead to believe that some infinite element is congruent to for 0 modulo 2 but 1 modulo 4, since one can write out the statement:

$$\forall x : M_{0,2}(x) \implies \neg M_{1,4}(x).$$

In the language of WS1S, which is true of G and hence in the theory of WS1S.

Definition. *Given a linear ordering L and a map $k : L \to \mathbb{K}$, define $\mathrm{PP}_{L,k}$ as:*

$$\mathrm{PP}_{L,k} := \{\mathrm{Pw}(k, \mathbf{x}, \mathbf{m}, \mathbf{p})\}.$$

And $H_{L,k}$ as the L-model with second order type $\mathrm{PP}_{L,k}$.

There's a little bit of work showing that $H_{L,k}$ is an L-model, which we will do in the proposition below.

As we shall see, since all of these elements are describable in terms of the endpoints of the intervals $x_0, \ldots, x_{n-1}$, they'll wind up existing in some form or other in every L-model of WS1S. Once we specify k, we have an exact set that must appear in our L-model of WS1S. What's more, the presence of just these elements is enough to satisfy our axioms, giving us a model of WS1S. This intuitively makes sense for our runnability schemes: given any deterministic automaton, on piecewise periodic input, we should expect a piecewise periodic run. If we have a long segment of periodic behavior of the input, after some amount of time settling in, the states of the automaton will also be periodic, of some period a factor of the original period times a number less than the number of states of our automaton. The time spent settling in is finite, so we just encode these all as pieces of period one.

Theorem 9. *$H_{L,k}$ is an L-model of WS1S.*

Proof. First, we need to show that $H_{L,k}$ is an L-model. Specifically, we need to show that it satisfies conditions (3) and (4). Condition (3) holds by construction. Suppose now that a set X is piecewise periodic and initial in every ω or ζ part. Assume there is some piece in the specification of X, $[x_i, x_{i+1})$ such that $[x_i, x_{i+1}) \cap X \neq [x_i, x_{i+1})$ and $[x_i, x_{i+1}) \cap X \neq \emptyset$. Consider the largest such piece. Looking in the last two periods of this piece, we can find an element x such that $x \notin X$ but $S(x) \in X$, contradicting that X is initial in the ω or ζ part in which x appears, contradicting our assumptions. As such, every periodic piece of X is either the complete interval or empty. If an empty interval were followed by a complete interval, then we would again have an $x \notin X$ with $S(x) \in X$, so it must be that X consists of a series of complete intervals followed by a series of empty intervals. That is, X is an interval of the form $\{x | x \leq \ell\}$ for ℓ the weak right endpoint of the largest nonempty interval.

As such, we've already checked axioms **(1.1)** through **(1.7)** and **(2.1)**.

(2.2): It is clear that singletons are piecewise periodic.

(2.3), (2.4): Let $x_0, \ldots, x_{n-1}$ be a common refinement of the endpoints of the intervals describing piecewise periodic sets X, Y. Then X and Y can also be described as piecewise periodic with pieces $[x_i, x_{i+1})$. On any one particular piece $[x_i, x_{i+1})$, X and Y will have certain periodic behaviors of periods m, m'. One can then verify that $X \cap Y$ and $X \setminus Y$ will have periodic behavior of period some factor of mm' on that particular piece. Hence, $X \cap Y$ and $X \setminus Y$ are piecewise periodic, and thus in our model.

(2.5) Given a piecewise periodic set X, we can find its largest element by looking in the last period of the last nonempty piece.

(3.A) Given a runnable automaton A, let A' be a deterministic automaton obtained by removing tuples from the transition relation of A. We will show that A' has a run on any suitable input $X_0, \ldots, X_{n-1}$, and thus so does A.

Given our inputs $X_0, \ldots, X_{n-1}$ piecewise periodic, we can find a common refinement of the endpoints of their pieces $y_0, \ldots, y_{r-1}$, and common periodicities $m_0, \ldots, m_{r-2}$. We can also make sure $y_0 = 0$ by possibly adding in new empty pieces to the beginning of the specifications of our inputs. Iterating through the intervals $[y_i, y_{i+1})$, do the following:

Step 1) Let $r_i : \{0, \ldots, m_i - 1\} \to 2^n$ describe the periodic behavior of the inputs such that for $w \in [y_i, y_{i+1})$,

$$w \in X_j \iff (r_i(w \bmod m_i))_j = 1.$$

Step 2) Consider the state of the automaton A just after entering the region

$[y_i, y_{i+1})$. If this is the first region, this is the start state. If this is not the first region, this was computed when we worked through the previous region. Call this state q. Let $m = m_i$.

Step 3) Consider the operation of one period (m characters) of input on states, giving us a map $f : Q \to Q$.

Step 4) Apply f to q repeatedly until a state is encountered that appeared before. This will give us a sequence of states:

$$q = q_0, q_1, \ldots, q_s, \ldots, q_{t-1}, q_t = q_s.$$

Note that t cannot be larger than $|Q|$, since there are only $|Q|$ distinct states.

Step 5a) In case $S^{mt}(y_i) \geq y_{i+1}$, append to our specification of each element of the run the interval $[y_i, y_{i+1})$ with period mt and periodic behavior just the appropriate behavior for a run of an automaton on the input in the interval $[y_i, y_{i+1})$. If this happens, skip step **6)** and go on to the next interval, remembering the state of the automaton after reading the final character.

Step 5b) Otherwise, append to our specification of each element of the run the interval $[y_i, S^{ms}(y_i))$ with period ms and periodic behavior just the appropriate behavior for the run of an automaton on the input in the interval $[y_i, S^{ms}(y_i))$ starting from state q.

Step 6) Now, append to our specification of each element of the run the interval $[S^{ms}(y_i), y_{i+1})$ with period $m(t-s)$ and periodic behavior the appropriate behavior for the run of an automaton on the input in the interval $[S^{ms}(y_i), S^{mt}(y_i))$, starting from state q_s. Since the state this interval starts in and the state immediately after the end are the same, this is a valid part of the run of A. Based on the remainder of y_{i+1} modulo $m(t-s)$, determine the state of the automaton after after reading the final character just before position y_{i+1}.

Repeat from step **2)** for each piece $[y_i, y_{i+1})$. This will construct the desired run of A' on input $X_0, \ldots, X_{n-1}$.

The proof for **(4.**A**)** for reverse-runnable A is identical, but in reverse. $\square$

Proposition 46. *Every L-model of WS1S contains $H_{L,k}$ as a subset of its second order objects for the appropriate k.*

Proof. Given an L-model of WS1S, we can determine k by looking at the congruence classes of elements of the form $(0, \ell)$. It then suffices to note that the elements $\mathrm{Pw}(k, \mathbf{x}, \mathbf{m}, \mathbf{p})$ are describable in terms of the elements $x_0, \ldots, x_{n-1}$. Since such elements exist for every increasing sequence $x_0, \ldots, x_{n-1}$ in the ground model G, they must exist in every nonstandard model as well. $\square$

Finally, we must ask the question: how many models have we actually found? Specifically, when is $H_{L,k}$ isomorphic to $H_{L',k'}$?

Definition. *Two congruence types* $\mathbf{s}, \mathbf{s}'$ *are said to be* shifts *if there is some k such that $s_n = s'_n + k \bmod n$ for each n.*

Proposition 47. *$H_{L,k}$ and $H_{L',k'}$ are isomorphic iff there is an isomorphism of linear orderings $f : L \to L'$ such that $k(\ell)$ and $k'(f(\ell))$ are shifts for each ℓ.*

Proof. ($\Rightarrow$): Suppose $H_{L,k}$ and $H_{L',k'}$ are isomorphic. Then there is a bijection $g : \omega + \zeta \cdot L \to \omega + \zeta \cdot L'$ and a bijection $h : \mathrm{PP}_{L,k} \to \mathrm{PP}_{L',k'}$. Recall that we defined an equivalence relation on $\omega + \zeta \cdot L$ by:

$$x \sim y :\equiv \exists n, m : S^n(x) = S^m(y).$$

Since g respects S, g respects these equivalence classes, giving us a bijection $g' : L \to L'$. g' must also respect $<$, since g respects $<$. This is the isomorphism f we're looking for. Suppose $f((0, \ell)) = (z, \ell')$. ℓ' will be $g'(\ell)$. By isomorphism, $(0, \ell)$ and (z, ℓ') will have the same modular equivalence classes, so $(0, \ell)$, and $(0, \ell')$ will have shifted modular equivalence classes, as desired.

($\Leftarrow$) is a straightforward construction of the isomorphism. $\square$

Proposition 48. *There are continuum many 1-models.*

Proof. Note that there are continuum many congruence types, and that the *shift* relation has countable equivalence classes. Hence there are continuum many congruences types up to shifting. The result then follows by the above proposition. $\square$

Now that we've shown that there are nonstandard models of WS1S, we would like a tool for classifying these nonstandard models.

4.5 The Tail-Head Lemma

Suppose we have a particular 1-model M of WS1S, and two second order objects X, X' from M which agree on the ω part. Since the symmetric difference of X, X' is an element of M, it has a least element. This means that there is some point in the ζ part such that before that point, X and X' agree. Put another way, the behavior of a second order element of M on the ω part determines an initial part of its behavior on the ζ part. In this section, we formalize and expand on this result.

Definition. *Given a linear ordering L, a* cut *is just a partition of L into two subsets (A, B) where A is an initial segment of L and B is a final segment of L. A cut (A, B) is said to be* infinite *if A has no greatest element and B has no least element.*

Proposition 49. *Any infinite cut of $\omega + \zeta \cdot L$ is of the form $(\omega + \zeta \cdot A, \zeta \cdot B)$ for some cut (A, B) of L.*

Proposition 50. *Given an L-model $(M_1, M_2, \in, S)$ of WS1S and an infinite cut (A, B) of M_1, $A, B \notin M_2$.*

Proof. Since $(M_1, M_2, \in, S)$ is an model of WS1S, every second order element must have a $<$-least and $<$-greatest element, which by assumption neither A nor B has. □

Proposition 51. *Let $(M_1, M_2, \in, S)$ be an L-model of WS1S. Let (A, B) be an infinite cut of M_1. Suppose $X, Y \in M_2$ are second order objects, and $x \in A$ and X and Y agree for all $y \in A$ with $y \geq x$. Then X and Y agree on some initial segment of B.*

Proof. Let $X \Delta Y$ denote the symmetric difference of X and Y. By closure under boolean operations, $X \Delta Y \in M_2$. Additionally, $X \Delta Y \setminus \{y | y \leq x\} \in M_2$, as this later set is the downward closure of the singleton $\{x\}$. This set, $X \Delta Y \setminus \{y | y \leq x\} \in M_2$, is either empty or has a $<$-smallest element. In either case, it doesn't contain elements up to some point in B. Put another way, X and Y agree up to that point in B. □

Proposition 52. *Let $(M_1, M_2, \in, S)$ be an L-model of WS1S. Let (A, B) be an infinite cut of M_1. Suppose $X, Y \in M_2$ are second order objects and $x \in B$ and X and Y agree for all $y \in B$ with $y \leq x$. Then X and Y agree on some final segment of B.*

Proof. The proof is a reverse version of the above. □

Definition. *Given an L-model with an infinite cut (A, B) on its first order part, we define equivalence relations $=_T^A$ (*tail equivalence*) and $=_H^B$ (*head equivalence*) on M_2, and more generally, on $\mathcal{P}(M_1)$ by:*

$$X =_T^A Y :\equiv \exists x \in A : \forall y \in A : y \geq x \implies y \notin X \Delta Y,$$
$$X =_H^B Y :\equiv \exists x \in B : \forall y \in B : y \leq x \implies y \notin X \Delta Y.$$

Where Δ denotes symmetric difference. Specifically, X and Y are tail equivalent if they agree on a final interval of A and are head equivalent if they agree on an initial interval of B.

We can now reinterpret our propositions from before as saying that given an infinite cut (A, B) the tail equivalence class of a second order object determines its head equivalence class, and the head equivalence class determines its tail equivalence class.

Definition. *Given an L-model with an infinite cut (A, B) on its first order part, we define its* tail-head function $\mathrm{TH}_{(A,B)}$*:*

$$\mathrm{TH}_{(A,B)} : M_2/(=_T^A) \to M_2/(=_H^B),$$

Such that, for any $X \in M_2$,

$$\mathrm{TH}_{(A,B)}([X]_{=_T^A}) = [X]_{=_H^B}.$$

Additionally, we may wish to view $\mathrm{TH}_{(A,B)}$ as a partial function:

$$\mathrm{TH}_{(A,B)} : \mathcal{P}(A)/(=_T^A) \to \mathcal{P}(B)/(=_H^B).$$

Following from our propositions above, we have:

Proposition 53. $\mathrm{TH}_{(A,B)}$ *is well-defined and 1-1.*

Definition. *We say that two sets X, Y are equal or congruent* mod finite *(written $X =^* Y$) if their symmetric difference is finite.*

Theorem 10. *Two second order elements X, Y of an L-model have the same head and tail equivalence classes for every infinite cut iff they are equal mod finite.*

Proof. ($\Rightarrow$): suppose we have two sets X, Y which have the same head and tail equivalence classes, but which are not equal mod finite. Consider those points in $Z = X \Delta Y$, which must be infinite. As mentioned in [9], every infinite linear ordering has an infinite ascending sequence or an infinite descending sequence. Assume Z with its inherited ordering has an infinite ascending sequence $x_0, \ldots, x_n, \ldots$ (the argument in the descending case is similar, although it is important to note that no infinite descending sequence can intersect the ω part). Split L into two parts, $A = \{\ell \in L : \exists n : x_n > \ell\}$, and $B = L \setminus A$ (which may be empty, but that isn't disallowed). It can easily be seen that (A, B) is an infinite cut. I assert that X and Y do not have the same $=^A_T$ equivalence class, since given any x it's not the case that X and Y agree on A after x by construction of A in terms of points where X and Y differ.

($\Leftarrow$) is immediate for each cut (A, B). There are only finitely many points to avoid, so one may easily avoid them. □

Finally, in this section we present the Tail-Head Lemma, which states that an L-model is determined by the set of its tail-head functions. But first, in order to prove this, we need a quick lemma about linear orderings in general:

Definition. *Given a linear ordering L, an infinite collection of intervals $\mathcal{I}$ is said to be a* cut covering family *if it satisfies:*

- *For each interior cut (A, B) (ie, $A, B \neq \emptyset$) of L, there is an interval I in $\mathcal{I}$ such that I contains some elements of A and some elements of B.*
- *For the exterior cuts $(\emptyset, L)$ and $(L, \emptyset)$, there are intervals I in $\mathcal{I}$ which are initial and nonempty and final and nonempty respectively.*

In either case, the interval is said to cover *the cut.*

Proposition 54. *For any linear ordering L and cut covering family $\mathcal{I}$ for L, there is a finite subcollection $\mathcal{I}' \subset \mathcal{I}$ having union all of L.*

Proof. Suppose we have an infinite cut covering family $\mathcal{I}$ for L. Consider S as the set of all $x \in L$ such that the interval $\{y \in L | y \leq x\}$ is a subset of a finite union of elements of $\mathcal{I}$. This set S is nonempty, as the interval covering the cut $(\emptyset, L)$ is initial, and thus alone covers a number of intervals of the form $\{y \in L | y \leq x\}$. S is also an initial interval of L, since if we can cover $\{y \in L | y \leq x\}$ and $w < x$, then the same set of intervals covers $\{y \in L | y \leq w\}$. Suppose S is not all of L. Consider the cut $(S, L \setminus S)$. This cut is covered by some interval I, which contains an element $s \in S$ and $t \notin S$. Add this interval to the finite cover for s and we get get a finite set of intervals covering up to t contradicting that $t \notin S$. Thus, S is all of L.

Finally, consider the final cut $(L, \emptyset)$. This is covered by a final interval I containing a point x. Thus all of L is covered by the finite set of intervals covering up to x together with I. □

Now we are prepared to prove the Tail-Head Lemma.

Theorem 11 (Tail-Head Lemma). *For given L, the set of second order elements of any L-model is determined by the set of its tail-head functions $\mathrm{TH}_{(A,B)}$ for every infinite cut*

(A, B) *of* $\omega + \zeta \cdot L$.

Proof. Suppose we have two L-models $H = (\omega + \zeta \cdot L, M_2, \in, S)$ and $H' = (\omega + \zeta \cdot L, M_2', \in, S)$, with the same tail-head functions for every infinite cut. Let $X \in M_2$. I claim that X is also in M_2'. This would suffice to prove the lemma.

For every infinite cut (A, B), consider an element Y in M_2' which agrees with X on its head and tail equivalence classes for that cut. In particular, there's some elements $a \in A$ and $b \in B$ such that $X \cap [a, b] = Y \cap [a, b]$. Note that $[a, b]$ is an element of M_2 and M_2' so $X \cap [a, b]$ is an element shared in common by H and H'. We collect these intervals as part of a cut covering family $\mathcal{I}$ with $[a, b]$ covering the cut (A, B).

For every finite cut (A, B), let a be the final element of A and b the initial element of B. Throw in the interval $[a, b]$ to $\mathcal{I}$ to cover (A, B). Note that $X \cap [a, b]$ is in both M_2 and M_2'.

For the cut $(\emptyset, \omega + \zeta \cdot L)$, we cover it by throwing the interval $\{0\}$ into $\mathcal{I}$. Note that $X \cap \{0\}$ is in both M_2 and M_2'.

For the cut $(\omega + \zeta \cdot L, \emptyset)$, let x be an upper bound for X. Cover our cut by throwing the interval $\{y | y > x\}$ into $\mathcal{I}$. Note that $X \cap \{y | y > x\} = \emptyset$ is in both M_2 and M_2'.

All together, this gives us a complete cut covering family $\mathcal{I}$. By the proposition above, there is a finite subcollection $\mathcal{I}' \subseteq \mathcal{I}$ having union all of $\omega + \zeta \cdot L$.

Take the union over all intervals I in $\mathcal{I}'$ of $X \cap I$. This is a finite union of elements of M_2', hence an element of M_2'. However, since all of these intervals together cover all of $\omega + \zeta \cdot L$, this union is X. Thus X is in M_2', as desired. $\square$

One necessarily asks the question: what sorts of tail-head functions are available? Taking the simplest example, one may ask what possible tail-head functions are available for the (ω, ζ) cut for 1-models of WS1S. In the next section, we propose a partial solution.

4.6 Countable 1-Models

Looking at the L-models $H_{L,k}$ that we constructed previously, we notice that the possible tail equivalence classes for the cut $(\omega, \zeta \cdot L)$ are just the periodic ones. Put another way, the set of all intersections of second order elements with the ω part is just the set of all eventually periodic subsets of ω. One may then wonder if, given a particular subset of ω, there is a model of WS1S that has second order elements, which, when intersected with the ω part, give us that particular subset. Or, more generally, if one has a suitably closed collection of subsets of ω, is there a model of WS1S for which that collection is exactly the set of intersections of second order elements with the ω part?

One can of course get whatever subsets one wants with a compactness argument:

Proposition 55. *Given a collection $\mathcal{A}$ of subsets of ω, there is a nonstandard model of WS1S such that for every $A \in \mathcal{A}$, there is a second order element X in the model whose intersection with ω is A.*

Proof. For each element $A \in \mathcal{A}$, add the second order constant symbol C_A to the language of WS1S. For each $A \in \mathcal{A}$ and $n \in \omega$, add either the formula $S^n(0) \in C_A$ or $S^n(0) \notin C_A$ as appropriate to the theory of WS1S. Together this produces a collection of formulas, any finite subset of which is satisfiable (if one has only finitely many restrictions on C_A, one may construct a finite set of natural numbers satisfying those restrictions). By the compactness theorem, it follows that there is a model of WS1S with constants $C_A | A \in \mathcal{A}$ for which the intersection $C_A \cap \omega = A$. $\square$

However, the proof of the compactness theorem makes no assurances on the sort of models being constructed. Following the proof in [8], we first complete the collection of formulas in question, and then perform a Henkin construction to get our model. If one of our sets A in $\mathcal{A}$ is unbounded, we will need to introduce a new first order object, call it ∞, to serve as an upper bound for C_A. Due to the arbitrariness of our choices in how we complete our collection of formulas, it may be the case that we wind up requiring that C_A and $C_{A'}$ agree on elements of the form $S^{-n}(\infty)$ for all n for A and A' with different tail equivalence classes. Every time this happens, we are required to add in a genuinely new first order element ∞' to represent an upper bound for $(C_A \Delta C_{A'}) \cap \{x : x \leq z\}$, as well as elements of the form $S^z(\infty)$ for integer z. As we can see, the L for these models can, unchecked, grow very large.

What if we wish to restrict our attention just to 1-Models? In case of countable $\mathcal{A}$, we can still find our nonstandard model. For clarity, we will refer to elements in the ζ portion of a 1-model as $\infty + z$ for appropriate z.

Proposition 56. *Any 1-model is determined by its tail-head function* $\mathrm{TH}_{(\omega,\zeta)}$.

Proof. $\omega + \zeta$ only has two infinite cuts, (ω, ζ) and $(\omega + \zeta, \emptyset)$. In the later case, we already know the tail head function. There is only one tail equivalence class, that of the empty set, since all second order elements are bounded, and it gets mapped to the only head equivalence class, which is trivial. □

Definition. *Given three sets X_-, X_0, X_1 partitioning ω, define:*

$$\mathrm{Bit}(X_-, X_0, X_1) = (Y_0, Y_1),$$

Such that $x \in Y_i$ if the largest $y \leq x$ which is in $X_0 \cap X_1$ is in X_i, or 0 if none exist.

Define:

$$\mathrm{RBit}(X_-, X_0, X_1) = (Y_0, Y_1),$$

Such that: $x \in Y_i$ if the smallest $y \geq x$ which is in $X_0 \cap X_1$ is in X_i, or 0 if none exist.

Definition. *Given sets X_σ for $\sigma \in S_n$ partitioning ω, define:*

$$\mathrm{Acc}_n(\mathbf{X}) = \mathbf{Y},$$

Where $\mathbf{Y}$ is also a collection indexed by elements of S_n, and $x \in Y_{\prod_{i<x} \sigma_i}$ where σ_i is such that $i \in Y_{\sigma_i}$.

Definition. *Given a collection of subsets $X_0, \ldots, X_{n-1}$ of the same set L, a* configuration *is an n-tuple of 0s and 1s. We say that the configuration* s *appears at $\ell \in L$ in $X_0, \ldots, X_{n-1}$ if:*

$$\ell \in X_i \iff s_i = 1.$$

We use the shorthand write **c** to $X_0, \ldots, X_{n-1}$ at position ℓ in L *to mean:*

- *For each i with $s_i = 1$, throw ℓ into X_i,*
- *For each i with $s_i = 0$, remove ℓ from X_i.*

Definition. *We adapt the bit storage, reverse bit storage, and accumulator on S_n automata from* Generating the Regular Functions and Krohn-Rhodes*:*

$$\mathrm{Bit} := (3, \{0,1\}, \{0\}, \delta, \{0,1\}),$$

Where δ is deterministic, with the b such that $\delta(a, \mathrm{s}, b)$ is given by the following table:

$\mathrm{s} \backslash a$	0	1
100	0	1
010	0	0
001	1	1
o.w.	0	0

Table 4.1: Transition Table for Bit

$$\mathrm{RBit} := (3, \{0,1\}, \{0,1\}, \delta, \{0\}),$$

Where the unique a such that $\delta(a, \mathrm{s}, b)$ is given by the following table:

$\mathrm{s} \backslash a$	0	1
100	0	1
010	0	0
001	1	1
o.w.	0	0

Table 4.2: Transition Table for RBit

Pick an enumeration of S_n.

$$\mathrm{Acc}_n := (n!, S_n, id, \delta, S_n),$$

Where δ is deterministic, with the b such that $\delta(a, \mathrm{s}, b)$ is given by:

- *$a \cdot \sigma$ where the only 1 in s is in position corresponding to σ,*
- *id if there isn't exactly one 1 in s.*

Note that these are meant to correspond to $\mathrm{Bit}, \mathrm{RBit},$ and Acc_n, respectively.

Proposition 57. *If an L-model satisfies* **(1.1)** *through* **(2.5)** *and* **(3.**Bit**)** *and* **(3.**Acc_n**)**, *then it satisfies* **(3.**A**)** *for arbitrary runnable A.*

Proof. It suffices to show that **(3.A)** holds for deterministic A. Recall from *Generating the Regular Functions and Krohn-Rhodes* that every map that takes a sequence of inputs and returns the run of A on those inputs with its last character removed can be written as a multivariable composition of character-wise maps and such truncated run maps for automata corresponding to Bit and Acc_n. One can check that this is just a statement about transition relations and holds even when the words being operated on are nonstandard.

As usual, one encodes words as partitions of the set of positions in the word. Via this encoding, character-wise maps become bitwise applications of bounded boolean functions. Specifically, if we have $f : \Sigma \to \Sigma'$, the translation of the character-wise application of f becomes:

$$f'(X_a | a \in \Sigma) = (Y_b | b \in \Sigma'),$$

Where:

$$Y_b = \bigcup_{a|f(a)=b} X_a.$$

The truncated run maps become the maps from inputs to runs for Bit and Acc_n, but with the final state removed. Since our L-model is closed under these maps (simply remove the final character of the runs by closure under finite modification), our L-model is closed under any of their multivariable compositions. This includes the map from inputs to runs for A with the final state removed. We can add in the final state by closure under finite modification, completing the proof of **(3.A)**. □

Proposition 58. *If an L-model satisfies* **(1.1)** *through* **(2.5)** *and* **(3.Acc_n)** *and* **(4.RBit)** *then it satisfies* **(4.A)** *for arbitrary reverse-runnable A.*

Proof. Recall that, up to character-wise maps, the transition relations for running Acc_n forward and Acc_n backwards are the same. The essential difference between saying that Acc_n has a forward run and that Acc_n has a reverse run is that in the forward run, the first state is specified, and in the backwards run, the last state is specified. However, by applying a character-wise premultiplication by the inverse of the last element of the forwards run of Acc_n, we get the reverse run of Acc_n.

The proof then continues similarly to the above proposition. □

Definition. *Given a countable set $\mathcal{W}$ of subsets of ω which:*

- *Contains $\emptyset, \omega$,*
- *Is closed under union and difference (and hence all bitwise maps),*
- *Is closed under finite modification,*
- *Is closed under* Bit, RBit *and* Acc_n *for every n.*

Let:

$$H^1_{\mathcal{W}} := (\omega + \zeta, M_{\mathcal{W}}, \in, S),$$

Where $M_{\mathcal{W}}$ is constructed via the following process:

Step 1) $\mathcal{W}$ is countable, so let $W_0, \ldots, W_n, \ldots$ be an enumeration of its elements. We will construct sets $Z_i \subseteq \mathbb{Z}$ with the intention of throwing in sets of the form $W_i \cup Z_i$ into our model. Begin our construction with the sets Z_i empty, and define a variable h, which we think of as the location of a *write head* as the integer 0.

Step 2) Iterate through $i \in \omega$, completing the following steps:

Step 3) Let C be the collection of configurations which appear infinitely often among $X_0, \ldots, X_i$. We will iteratively make sure that when beginning this step, the write head will be located at a configuration $\mathbf{c}$ in $Z_j | j < i$ such that the configuration $\mathbf{c}$ appears infinitely often among $X_0, \ldots, X_{i-1}$. Note that this is trivially true in the case of $i = 0$.

Step 4) Since the configuration $\mathbf{c}$ appears infinitely often among $X_0, \ldots, X_{i-1}$ then either $\mathbf{c}^\frown 0$ or $\mathbf{c}^\frown 1$ appears infinitely often among $X_0, \ldots, X_i$. In the first case, do nothing. In the second case, add h (the location of the write head) to Z_i, so that either way the write head is now located at a configuration in $Z_j | j \leq i$ which appears infinitely often among $X_0, \ldots, X_i$.

Step 5) Iterating through configurations $\mathbf{c}$ in C (note that C must be finite), complete the following steps:

Step 6) Letting $\mathbf{c}'$ be the configuration at the current write head location, locate a sequence of configurations

$$\mathbf{c} = \mathbf{c}_0, \ldots, \mathbf{c}_{k-1} = \mathbf{c}',$$

Such that:

- Each $\mathbf{c}_j$ is from C,
- The sequence of configurations occurs consecutively within $X_0, \ldots, X_i$.

We can find such a configuration by locating an instance of the configuration $\mathbf{c}$ beyond the point in ω where those configurations which appear only finitely often among $X_0, \ldots, X_i$ finish appearing, and then locating an instance of the configuration $\mathbf{c}'$ which appears after that. Our sequence is the sequence of configurations which appear between those instances of $\mathbf{c}$ and $\mathbf{c}'$ inclusive.

Step 7) Write the configurations $\mathbf{c}_0, \dots, \mathbf{c}_{k-1}$ to locations $h-k+1, \dots, h$ respectively. Note that configuration $\mathbf{c}_{k-1}$ is already present at position h. Move the write head to position $h-k+1$.

Step 8) If there are still more $\mathbf{c} \in C$ which we haven't iterated through, move on to the next $\mathbf{c} \in C$ and continue from step **6)**. Otherwise go on to step **9)**.

Step 9) Move on to the next $i \in \omega$, continuing from step **3)**. Complete this for all $i \in \omega$, then go on to step **10)**.

Step 10) Throw into our set $M_{\mathcal{W}}$ every set of the form $W_i \cup Z_i'$ for $i \in \omega$ and Z_i' a finite modification of Z_i.

This completes our construction.

Proposition 59. *When this construction has completed, a configuration appears infinitely often within $W_0, \dots, W_j$ iff it appears initially often within $Z_0, \dots, Z_j$ for any j*

Proof. (⇒): Note that for each $a < b$, if the configuration $\mathbf{c}$ appears infinitely often within $W_0, \dots, W_a$ then there is an extension of $\mathbf{c}$ which appears infinitely often within $W_0, \dots, W_b$, by the infinite pigeonhole principle. Thus, for each iteration referred to in step **2)** for $i \geq j$, we write $\mathbf{c}$ to $Z_0, \dots, Z_j$ at least once. Since there are infinitely many such iterations, this completes our proof.

(⇐): Suppose the configuration $\mathbf{c}$ appears infinitely often within $Z_0, \dots, Z_j$. Then it must be that for each iteration i referred to in step **2)**, we write configuration $\mathbf{c}$ to $Z_0, \dots, Z_j$, which means that some extension of $\mathbf{c}$ appears infinitely often among $W_0, \dots, W_i$. From this we can conclude that $\mathbf{c}$ appears infinitely often among $W_0, \dots, W_i$. □

Proposition 60. *Let $X_0, \ldots, X_{n-1}$ be arbitrary elements of $M_{\mathcal{W}}$. Then a configuration appears infinitely often within the ω part of $X_0, \ldots, X_{n-1}$ iff it appears initially often within the ζ part of $X_0, \ldots, X_{n-1}$.*

Proof. ($\Rightarrow$): Note that there is some r such that $X_0, \ldots, X_{n-1}$ all come from sets of the form $W_i \cup Z'_i$ for $i < r$. Suppose a configuration **c** appears infinitely often within the ω part of $X_0, \ldots, X_{n-1}$. Then some suitable extension of **c** appears infinitely often within $W_0, \ldots, W_{r-1}$. This extension appears infinitely often within $Z_0, \ldots, Z_{r-1}$. Thus it appears infinitely often among $Z'_0, \ldots, Z'_{r-1}$. Thus **c** appears infinitely often within the ζ part of $X_0, \ldots, X_{n-1}$.

($\Leftarrow$) is the reverse of ($\Rightarrow$). $\square$

Proposition 61. *Given two elements X, Y in $M_{\mathcal{W}}$, if X and Y eventually agree on their ω parts, then they initially agree on their ζ parts.*

Proof. Suppose X is $W_a \cup Z'_a$ and Y is $W_b \cup Z'_b$. Wlog, assume $a < b$. Then configurations which appear infinitely often among $W_0, \ldots, W_b$ agree in their a and bth coordinates. Hence configurations which appear infinitely often among $Z_0, \ldots, Z_b$ agree in their a and bth coordinates. Hence Z_a and Z_b agree initially (only the empty configuration appears after 0 in the ζ part). Hence Z'_a and Z'_b agree initially. $\square$

Proposition 62. *$M_{\mathcal{W}}$ is closed under finite modification.*

Proof. Suppose we have an X in $M_{\mathcal{W}}$ and wish to modify whether it contains x or not. If x is in the ζ part, this holds by construction. Otherwise, X is of the form $W_i \cup Z'_i$. Let W_j be W_i modified appropriately at x (recall that $\mathcal{W}$ was closed under finite modifications). Then $Y = W_j \cup Z_j$ is in our model.

Since X and Y eventually agree on their ω parts, they initially agree on their ζ parts. Thus Z_j is a finite modification of Z'_i so $W_j \cup Z'_i$ is in our model, which is the desired modification of X. $\square$

Proposition 63. *Every set in $M_{\mathcal{W}}$ has a largest element.*

Proof. Suppose we have an $X = W_i \cup Z'_i$. Recall that Z_i had upper bound $\infty + 0$. If Z'_i is empty, one can argue that W_i does not contain infinitely many elements, and so has a maximal element which is also the maximal element of X. If Z'_i is nonempty, since it is a finite modification of a set with upper bound $\infty + 0$, it has a maximal element, which is also the maximal element of X. $\square$

Proposition 64. *Given elements $X_0, \dots, X_{n-1}$ of $M_{\mathcal{W}}$, if the configurations $\mathbf{c}, \mathbf{c}'$ appear consecutively coinitially in the ζ part among $X_0, \dots, X_{n-1}$, then they appear consecutively in the ω part.*

Proof. Using the same sorts of configuration expansion and finite modification arguments as in previous propositions, it suffices to prove this for $X_i = W_i \cup Z_i$ for $i < n$. By construction, anywhere in the ζ part before the position where the write head was located at the beginning of the iteration for n, any consecutive pair of configurations among $Z_0, \dots, Z_{n-1}$ will come from the same sequence of configurations as in step **6)**, which was copied directly from $W_0, \dots, W_{n-1}$. $\square$

Theorem 12. *Given a countable set $\mathcal{W}$ of subsets of ω which:*

- *Contains $\emptyset, \omega$,*
- *Is closed under union and difference (and hence all bitwise maps),*
- *Is closed under finite modification,*

- *Is closed under* Bit, RBit *and* Acc_n *for every* n.

It is the case that $H^1_{\mathcal{W}}$ *is a model of WS1S.*

Proof. First, we need to show that $H^1_{\mathcal{W}}$ is a 1-model. Specifically, we need to show that it contains sets of the form $\{x|x < \ell\}$, and we need to show that every predecessor closed set is of that form.

I claim that $M_{\mathcal{W}}$ contains the empty set, since $\mathcal{W}$ contained the empty set, so some X in $M_{\mathcal{W}}$ has empty ω part. But by a previous proposition, the configuration $\mathbf{0}$ appears infinitely often in the ζ part of X and the configuration $\mathbf{1}$ does not appear infinitely often in the ζ part of X. Thus X is of the form $\emptyset \cup Z'_i$ for Z'_i a finite set. Since $M_{\mathcal{W}}$ is closed under finite modifications, the empty set is in $M_{\mathcal{W}}$.

Since $M_{\mathcal{W}}$ was closed under finite modifications, it contains all sets of the form $\{x|x < \ell\}$ for $\ell \in \omega$.

I claim the $M_{\mathcal{W}}$ also contains the interval $[0, \infty{+}0]$. Recall that $\mathcal{W}$ contained ω, so some X in $M_{\mathcal{W}}$ has full ω part. By a previous proposition, the configuration $\mathbf{1}$ appears infinitely often in the ζ part of X and the configuration $\mathbf{0}$ does not appear initially often in the ζ part of X. Thus X contains and so is a finite modification of some interval of the form $[0, z + \infty]$. By closure under finite modification, all intervals of the form $\{x|x < \ell\}$ are in $M_{\mathcal{W}}$ for ℓ in the ζ part.

Next, we show that the only sets in $M_{\mathcal{W}}$ which are predecessor closed are of the form $\{x|x < \ell\}$. Suppose we have a predecessor-closed set X. If it doesn't contain any elements from the ζ part, it is a predecessor closed subset of ω, thus of the desired form. If it does contain any elements from the ζ part, it contains an initial segment of them. Thus, it contains a final segment of ω. By downward closure, it thus contains all of ω. One can then argue that it is of the desired form by looking at its maximal element.

As such, $H^1_{\mathcal{W}}$ is a 1-model, and so satisfies axioms **(1.1)** through **(1.7)** and **(2.1)**. Now we check the rest:

(2.2) was previously proved.

(2.3) Suppose we have two sets $W_i \cup Z'_i$ and $W_j \cup Z'_j$ from $M_{\mathcal{W}}$. Since $\mathcal{W}$ was closed under boolean operations, it contains some $W_k = W_i \cup W_j$. The only configurations that show up in the ω part of the tuple $W_i \cup Z'_i, W_j \cup Z'_j, W_k \cup Z_k$ are of the form $\mathbf{000}, \mathbf{011}, \mathbf{101}$, and $\mathbf{111}$. Thus these are the only configurations which show up initially in their ζ parts. By upper bounding, the only the configuration $\mathbf{000}$ shows up finally in their ζ parts. By finite modification of Z_k to get Z'_k, we can fix those finitely many positions where these configurations do not appear, thus making sure these are the only configurations which show up in the tuple $W_i \cup Z'_i, W_j \cup Z'_j, W_k \cup Z'_k$. That is to say, $W_k \cup Z'_k$ is the desired union.

(2.4) is similar.

(2.5) was proved in a previous proposition.

In order to prove axiom **(3.A)** we will first prove **(3.**Bit**)** and **(3.**Acc_n**)**.

(3.Bit**)**: Suppose we are given a triple of inputs $W_i \cup Z'_i, W_j \cup Z'_j, W_k \cup Z'_k$. Since $\mathcal{W}$ was closed under the Bit map, there is some (W_g, W_h) which has the correct start state and satisfies the transition relation for Bit at all positions in ω for inputs $W_i \cup Z'_i, W_j \cup Z'_j, W_k \cup Z'_k$. Now look at Z_g and Z_h. By an above proposition, any consecutive pair of configurations that appear initially in the ζ part among

$W_i \cup Z'_i, W_j \cup Z'_j, W_k \cup Z'_k, W_g \cup Z_g, W_h \cup Z_h$ will satisfy the transition relation for Bit. Thus, by finite modification, we can find Z'_g, Z'_h such that $(W_g \cup Z'_g, W_h \cup Z'_h)$ is the desired run of Bit on input $W_i \cup Z'_i, W_j \cup Z'_j, W_k \cup Z'_k$.

(3.Acc_n**)**: Similar to the above but with Acc_n.

Thus, as proved in an earlier proposition, **(3.**A**)** holds for every runnable A.

In order to prove **(4.**A**)** we will first prove **(4.**RBit**)** (we just proved **(3.**Acc_n**)**).

(4.RBit**)**: Suppose we are given a triple of inputs $W_i \cup Z'_i, W_j \cup Z'_j, W_k \cup Z'_k$. We divide the argument up into two cases: one in which $W_i \cup Z'_i$ has the full head and tail equivalence classes and one in which it doesn't.

Case 1) If $W_i \cup Z'_i$ has the full head and tail equivalence classes, one can easily verify that the desired run (Y_0, Y_1) will have one of Y_0 or Y_1 in the full head and tail equivalence classes and one in the empty head and tail equivalence classes. Thus one is a finite modification of the empty set, and the other is a finite modification of the interval $[0, \infty + 0]$.

Case 2) Otherwise, since $\mathcal{W}$ was closed under the RBit map, there is some (W_g, W_h) which satisfies the transition relation for RBit at all positions in ω for inputs $W_i \cup Z'_i, W_j \cup Z'_j, W_k \cup Z'_k$. Now look at Z_g and Z_h. By an above proposition, any consecutive pair of configurations that appear initially in the ζ part among $W_i \cup Z'_i, W_j \cup Z'_j, W_k \cup Z'_k, W_g \cup Z_g, W_h \cup Z_h$ will satisfy the transition relation for RBit. Thus, by finite modification (and we need that either $W_j \cup Z'_j$ or $W_k \cup Z'_k$ make appearances in the ζ part to do this) we can find Z'_g, Z'_h such that $(W_g \cup Z'_g, W_h \cup Z'_h)$ is the desired run with the correct end state of RBit on input $W_i \cup Z'_i, W_j \cup Z'_j, W_k \cup Z'_k$.

Thus, as proved in an earlier proposition, **(4.A)** holds for every reverse-runnable A. □

Now that we have a technique for constructing 1-models with interesting tail-head functions, we would like a technique for moving these tail-head functions around. In the next section, we describe as simple cut-and-paste technique for producing new models of WS1S from old.

4.7 Cut-And-Paste Models

In this section, we describe a procedure for gluing together a number of separate models of WS1S to produce a new model of WS1S.

Proposition 65. *In any model of WS1S, if X is a nonempty second order elements of the model, I is an interval with least element, and $X \subseteq I$, and:*

$$\forall x : S(x) \in X \implies (x \in X \vee x \notin I),$$

Then X is of the form $[a, b]$ for a the minimal element of I and b is some element of I.

When the hypothesis holds, we say that X is nonempty and predecessor closed within *I.*

Proof. Let a be the minimal element of I. I claim that $X \cup [0, a)$ will be predecessor closed, and thus of the form $[0, b]$ for b the maximal element of X. Thus X will be of the form $[a, b]$. □

Proposition 66. *In any model of WS1S, if (n, Q, I, δ, Q) is a runnable finite automaton, and $X_0, \ldots, X_{n-1}$ are second order elements of our model, for any ℓ there is a* *delayed run of our finite automaton, starting at ℓ and satisfying the transition relation δ. Specifically, there are $Y_0, \ldots, Y_{|Q|-1}$ such that:*

- *The Y_q partition the interval $[\ell, S(\max \bigcup \mathbf{X})]$,*
- *$\ell \in Y_q$ for some $q \in I$,*
- *If $\ell \leq x \leq \max \bigcup \mathbf{X}$, and $x \in Y_q, x \in X_{\mathbf{s}}, S(x) \in Y_{q'}$, then $\delta(q, \mathbf{s}, q')$.*

Proof. One can prove this is true of the standard model of WS1S by simple induction. Since we can express this fact in the language of WS1S, it must be true of any nonstandard model. $\square$

Definition. *A* cut-and-paste scheme *is a tuple:*

$$(L, I_0, \ldots, I_{n-1}, H_0, \ldots, H_{n-1}, J_0, \ldots, J_{n-1}, f_0, \ldots, f_{n-1}),$$

Where:

- *L is a linear ordering of the form $\omega + \zeta \cdot L'$,*
- *The $I_0, \ldots, I_{n-1}$ form a partition of L into intervals,*
- *H_i is an L_i-model of WS1S, for some linear order L_i,*
- *J_i for $i < n-1$ is an interval in the second order part of H_i,*
- *J_{n-1} is a final interval of the first order part of H_{n-1},*
- *$f_i : I_i \to J_i$ is an order preserving bijection.*

Definition. *Given a cut-and-paste scheme $(L, \mathbf{I}, \mathbf{H}, \mathbf{J}, \mathbf{f})$, define the* resolution *of the cut-and-paste scheme as the L-model with second order part:*

$$W := \{\bigcup_i f^{-1}(X_i) | X_i \subseteq J_i\}.$$

Where the X_i are required to be second order elements of their respective H_i.

Theorem 13. *Given a cut-and-paste scheme $(L, \mathbf{I}, \mathbf{H}, \mathbf{J}, \mathbf{f})$, its resolution $(\omega + \zeta \cdot L, W, \in, S)$ is an L-model of WS1S.*

Proof. For $i < n - 1$, we note that since the J_i are intervals in the second order part of H_i, they have least and greatest elements, and thus since f_i is an order preserving bijection, the I_i have least and greatest elements as well. This leaves I_{n-1} (since $\mathbf{I}$ form a partition) to have a least element but no greatest element.

First, we need to show that $(\omega + \zeta \cdot L, W, \in, S)$ is actually an L-model. As usual, we just need to check that every interval of the form $\{x|x < \ell\}$ is in W and that the only sets X in W which when intersected with each ω or ζ part form an initial interval of that part are of the form $\{x|x < \ell\}$.

Given an ℓ, there is some interval I_i in the cut-and-paste scheme containing ℓ. Since H_i is a model of WS1S, it contains the interval $[\min(J_i), f_i(\ell))$. What's more, H_j contains the interval J_j for $j < i$ (since such j cannot be $n - 1$). One simply verifies then that:

$$\begin{aligned}[0, \ell) =& \bigcup_{j<i} f^{-1}(J_j) \cup f^{-1}([\min(J_i), f_i(\ell))) \\ =& \bigcup_{j<i} I_j \cup [\min(I_i), \ell)\end{aligned}$$

Now, suppose that $X \in W$ which when intersected with each ω or ζ part forms an initial interval of that part. Then I claim that for each i, $X \cap I_i$ forms an initial interval of I_i. Note that X must be predecessor closed. Recall that $X \cap I_i$ was the image under the order preserving bijection f of some second order element X_i in H_i, which has to be a subset of J_i. Thus, X_i is predecessor closed in J_i, and thus by our lemma above, some initial interval of J_i. It then follows by our bijection that $X \cap I_i$ forms an initial interval of I_i. Combine this with the fact that if X contains the least element of I_i then by predecessor closure, it contains the greatest element of I_{i-1}, and we can see that X must be an interval of the form $\{x|x < \ell\}$.

Thus, the resolution of the cut-and-paste scheme is an L-model, and thus satisfies **(1.1)** through **(1.7)** and **(2.1)**.

(2.2): Suppose we have an $x \in \omega + \zeta \cdot L$. Then it fell into some interval I_i. Let:

$$X_j = \begin{cases} \{f(x)\} & j = i \\ \emptyset & j \neq i, \end{cases}$$

Then I claim that

$$\{x\} = \bigcup_i f^{-1}(X_i).$$

(2.3): Suppose we have two sets A, B, with:

$$A = \bigcup_i f^{-1}(A_i),$$
$$B = \bigcup_i f^{-1}(B_i),$$

With A_i, B_i second order elements of H_i which are subsets of J_i. Then I claim that:

$$A \cup B = \bigcup_i f^{-1}(A_i \cup B_i).$$

With $A_i \cup B_i$ second order elements of H_i which are subsets of J_i.

(2.4): The proof is the same as the proof for **(2.3)**.

(2.5): Suppose we have a set X with:

$$X = \bigcup_i f^{-1}(X_i),$$

With X_i second order elements of H_i which are subsets of J_i. Let i be the largest such that X_i is nonempty. Then X_i has a largest element. The image of this largest element is the largest element of X within I_i, and thus the largest element of X overall.

(3.A) Suppose we have runnable automaton $A = (m, Q, I, \delta, Q)$, and suitable input $X_0, \ldots, X_{m-1}$, with:

$$X_j = \bigcup_i f^{-1}(X_{j,i}).$$

With $X_{j,i}$ second order elements of H_i which are subsets of J_i.

Iterating through possible values of i do the following:

Step 1) Determine the state of A going into the beginning of I_i. If $i = 0$ this is the start state. Otherwise, we computed this at the end of the previous iteration.

Step 2) Use the proposition above to find a run of A starting at the minimal element of J_i on input $X_{0,i}, \ldots, X_{m-1,i}$, running up to either the largest element of J_i or the image of the largest element of $\bigcup \mathbf{X}$, whichever is smaller (at least one will exist).

Step 3) If we ran up to the largest element of $\bigcup \mathbf{X}$, glue together the images under f^{-1} of the found runs, I claim that this is the desired run. Otherwise, add one to i and go back to step **1)**, remembering the state we wound up in at the end.

(4.A) The argument is similar to the above. Note that if we're looking in a model of WS1S for a run of a reverse-runnable automaton A on an interval $[a, b]$ which ends in a particular state q, we simply use axiom **(4.A)** on inputs $X_0 \cap [0, S^{-1}(b)], \ldots, X_{m-1} \cap [0, S^{-1}(b)]$.

Since all of the axioms are satisfied, the resolution is a model of WS1S. □

Note that our initial restrictions on $\mathcal{W}$ were necessary of any projection of the second order part of a model of WS1S to its ω part.

4.8 Conclusion and Further Research

This paper makes significant strides towards the complete classification of nonstandard models of WS1S. In L-models we found a constructive canonical form for a representative of each isomorphism class of nonstandard model, and showed that for every linear ordering we could possibly expect as the first order part of a nonstandard model of WS1S (those of the form $\omega + \zeta \cdot L$), there is some nonstandard model of WS1S with that first order part when ordered by that model's interpretation of $<$. Tail-head functions gave us a language for specifying our L-models precisely, allowing us to completely capture all of the relevant information about a nonstandard model of WS1S.

Additionally, we provided a technique for producing a large variety of 1-models, and then a technique for gluing together pieces of nonstandard models of WS1S to produce new models. Combining these two, we could also produce a large variety of n-models, where n is the natural linear ordering on $\{0, \dots, n-1\}$.

This is a very promising start, but it is missing a few key pieces that will be necessary for any complete classification of 1-models. First, as we can see in the example of $H_{1,k}$ for various k, the projection of the second order part of a 1-model of WS1S to its ω part is not enough to uniquely determine the model. There may be many ways of completing those projections to a full 1-

model of WS1S. As such, the question of what sets of bounded subsets of $\mathbb{Z}$ can be completed backwards to produce a 1-model of WS1S is technically open, although it is likely that by allowing reversal in our cut-and-paste schemes this question can be resolved quickly.

One would also like to be able to extend this result to study the uncountable sets of subsets of ω that can be extended to 1-models of WS1S. The construction in our proof has an iterative building nature that closely resembles what one gets if one unpacks certain forcing constructions. It is possible that our argument may be obfuscated by translating it into a forcing proof, which may yield insights into how to extend it for uncountable collections $\mathcal{W}$.

Additionally, we haven't yet answered the question of what sets of subsets of ω can be completed to produce an L-model of WS1S for L without initial element. If L does have an initial element, we can use cut-and-paste arguments to turn L-models into 1-models with the same second order projection to their ω parts and vice versa. We cannot do such a cut-and-paste argument for L without initial element, although it is likely that a modification of the original construction would suffice. One gets a sense from how the Henkin construction freely adds new first-order elements as necessary that perhaps the 1-model case is the hardest.

If we look at those sets of subsets of ω which can be completed to produce a 1-model of WS1S, we see that there's some notion of regular function that they need to be closed under. This leads us to define a partial ordering $\leq_A$ on tuples of subsets of ω (which we can encode as ω sequences of characters from some finite alphabet) where $\mathbf{X} \leq_A \mathbf{Y}$ if $\mathbf{X}$ is the output of some regular function on input $\mathbf{Y}$. It is suspected that understanding the structure of this partial ordering will yield insights into the class of all 1-models of WS1S.

And of course, all of this work can be done over again in at least three other cases:

- S1S, the theory of ω and arbitrary subsets of ω with $\in$ and S,
- WS2S, the theory of finite strings of 0s and 1s and finite sets of such strings, with $\in$ and S_0, S_1 which tack a 0 or 1 onto the end of a string.
- S2S, the theory of finite strings of 0s and 1s and arbitrary sets of such strings, with $\in$ and S_0, S_1.

Initial speculation suggests that the S1S case should be fairly easy. The most technical portion of each proof will likely be showing that the standard construction for complementing automata over these structures works correctly.

APPENDIX A

APPENDIX A: LIST OF AXIOMS

(1.1)	$\forall x, y, z : x < y \wedge x < z \implies x < z$	($<$ is transitive)
(1.2)	$\forall x, y : x < y \implies \neg(y < x)$	($<$ is antisymmetric)
(1.3)	$\forall x, y : x < y \vee x = y \vee y < x$	($<$ is total)
(1.4)	$\forall x : S(x) > x$	(S is strictly increasing)
(1.5)	$\forall x : \nexists y : x < y < S(x)$	(S is a successor)
(1.6)	$\exists o : \forall x : o \leq x$	($<$ has minimal element)
(1.7)	$\exists o : \forall x : x \neq o \implies \exists y : S(y) = x$	(S has image $\omega \setminus \{0\}$)
(2.1)	$\forall X, Y : (\forall x : x \in X \Leftrightarrow x \in Y) \Rightarrow X = Y$	(Extensionality)
(2.2)	$\forall x : \exists X : \forall y : y \in X \Leftrightarrow y = x$	(Singleton)
(2.3)	$\forall X, Y : \exists Z : \forall x : x \in Z \Leftrightarrow (x \in X \vee x \in Y)$	(Union Closure)
(2.4)	$\forall X, Y : \exists Z : \forall x : x \in Z \Leftrightarrow (x \in X \wedge x \notin Y)$	(Difference Closure)
(2.5)	$\forall X \neq \emptyset : \exists x : x \in X \wedge \forall y : y \in X \Rightarrow y \leq x$	(Maximal Element)
(3.A**)**	$\forall X_0, \ldots, X_{n-1} : \Phi_{(n,Q,I,\delta,Q)}(X_0, \ldots, X_{n-1})$	(A Runnable)
(4.A**)**	$\forall X_0, \ldots, X_{n-1} : \Phi_{(n,Q,Q,\delta,F)}(X_0, \ldots, X_{n-1})$	(A Reverse-Runnable)

BIBLIOGRAPHY

[1] Büchi, J. R. (1990). On a Decision Method in Restricted Second Order Arithmetic. *The Collected Works of J. Richard Büchi*, 425-435.

[2] Downey, R. G., & Fellows, M. R. (1999). *Parameterized complexity*. New York: Springer.

[3] Egri-Nagy, A., & Nehaniv, C. (2005). Algebraic Hierarchical Decomposition of Finite State Automata: Comparison of Implementations for Krohn-Rhodes Theory. *Implementation and Application of Automata Lecture Notes in Computer Science*, 315-316.

[4] Egri-Nagy, A., & Nehaniv, C. (2006). Making Sense of the Sensory Data Coordinate Systems by Hierarchical Decomposition. *Lecture Notes in Computer Science Knowledge-Based Intelligent Information and Engineering Systems*, 333-340.

[5] Ginzburg, A. (1968). *Algebraic theory of automata*. New York: Academic Press.

[6] Kleene, S. C.: 1956, 'Representation of Events in Nerve Nets and Finite Automata', in C. E. Shannon and J. McCarthy (eds.), *Automata Studies*, Princeton University Press, Princeton, NJ, pp. 3-42.

[7] Krohn, K., & Rhodes, J. (1965). Algebraic Theory of Machines. I. Prime Decomposition Theorem for Finite Semigroups and Machines *Transactions of the American Mathematical Society*, 450-450.

[8] Marker, D. (2002). *Model theory: An introduction*. New York: Springer.

[9] Rosenstein, J. G. (1982). *Linear orderings*. New York: Academic Press.

[10] Siefkes, D. (1978). An axiom system for the weak monadic second order theory of two successors. *Israel J. Math. Israel Journal of Mathematics, 30*(3), 264-284.