Formal Languages and Groups as Memory:

An Exposition of the Main Step

NAME REDACTED

Abstract

The primary goal of this paper is to present an exposition of the main step of Kambites proof of the Chomsky-Schutzenberger representation theorem as well as a formal construction of M-automata and Dyck languages in the LEAN theorem prover. Formalization, in this case, serves as a tool to aid in exposition. To make the exposition as clear as possible, we give proofs in a human-readable format equivalent to the formalized proofs (the reader may find the formalized proofs here : `https://github.com/koly777/chomsky-schutzenberger-lean`).

We only present proofs that are directly relevant and/or showcase interesting content.

Contents

1 Preliminaries

It would be helpful for the reader if they are familiar with the basics of automata theory, though not strictly necessary for this paper. This section collects some results about solving equations in the free monoid over some alphabet X. Recall that the *free monoid over* X, denoted X^* consists of all words whose letters come from some alphabet X. We will denote by ϵ, the empty word consisting of no letters.

In particular we will make use of the following lemmas frequently. We note that these lemmas are already present in the LEAN library, and so we give them without proof.

Lemma 1 *For words $u, v, u', v \in X^*$, $uv = u'v'$ iff there exists w such that $u' = uw$ and $u = wv'$ or there exists a w' such that $u = u'w'$ and $v' = w'v$.*

Lemma 2 *For words $u, v, u' \in X^*$ and $x \in X$, we have that $uv = xu'$ iff $u = \epsilon$ and $v = xu'$ or there exists a w such that $u = xw$ and $u' = wv$.*

2 M-automata

M-automata serve as a generalisation of certain classes of blind automata equipped with some storage mechanism. An example of such blind automata is the class of blind k-counter automata studied by Sheila Greibach in her paper 1978 paper “Remarks on Blind and Partially Blind One-Way Multicounter Machines” which was helpful in constructing a formal definition of “blindness”.

An M-automaton over a monoid M is a non-deterministic finite state machine augmented with a register that may store a given element of the monoid. At any point in its operation, it may multiply the contents of the register on the right by an element of the monoid as it transitions from state to state. The automaton may not use the contents of the register to transition into another state, and this is what is mean when M-automata are described as being “blind”.

The automaton is initialized with the contents of the register set to $1 \in M$. A word is accepted by the automaton if after reading the word, the contents of the register returns to 1.

More formally an M-automaton over a monoid M consists of a finite alphabet X, a finite set of states Σ, a start state q_0, a set of final states $Q \subseteq \Sigma$ and a transition function $\delta : \Sigma \times (X \cup \{\epsilon\}) \to \mathcal{P}(\Sigma \times M)$. Thus the function δ takes in a state and a letter of the alphabet X (or the empty word ϵ) and returns a set of pairs (q, m) where $q \in \Sigma$ and

$m \in M$. We now define the operation of reading a word. We define the binary relation $\vdash$ on $\Sigma \times (X \cup \{\epsilon\})^* \times M$ by

$$(\sigma, aw, m) \vdash (\sigma', w, n)$$

where $a \in (X \cup \{\epsilon\})$ and $w \in (X \cup \{\epsilon\})^*$, if there exists an m' such that $(\sigma', m') \in \delta(\sigma, a)$ and $n = m \cdot m'$.

This relation corresponds to the process of reading a letter a or if $a = \epsilon$, then it corresponds to transitioning on the empty string. Denote by $\vdash^*$, the reflexive and transitive closure of $\vdash$. We say that a word w is *accepted* if $(q_0, w, 1) \vdash^* (q_f, \epsilon, 1)$ holds and $q_f \in Q$.

2.1 A Formal Construction of M-Automata

We formalize the class of M-automata as a structure in LEAN which we call `bRMA` [1]for "blind Register Monoid Automata". The parameters of this structure consists of a type M which has a monoid instance, an alphabet type α, a type of states σ, and an indexing type ι which indexes into the elements of M. We allow that each of these types be infinite, and finite type instances for α, σ and ι must be provided for true M-automata.

The fields of `bRMA` consists of a transition function which we call `step` of type $\sigma \to$ `option` $\alpha \to$ `set` $(\sigma \times \iota)$. We note that the option type consists of elements of the form `none` (which we take to mean the empty string ϵ) or `some` (a) where a is an element of type α corresponding to a letter a. Thus `step` is a function which takes in a state of type σ, a letter or the empty string and returns a set consisting of pairs whose elements are of type $\sigma \times \iota$. The next field is called `start` which is an element of type σ, this is the designated start state. The field `accept` is an element of type `set` σ corresponding to the final states and finally the field `to_monoid` is a function that takes in an element of the indexing type ι and produces an element of M.

We then define a relation which we call `cstep` (corresponding to $\vdash$) which takes in two triples of type $\sigma \times list($`option` $\alpha) \times M$ and returns a proposition (we note that a string of letters in $(X \cup \{\epsilon\})^*$ is modelled as a list whose elements come from the option type over α). The relation states that $(s, w_1 \cdots w_n, m)$ is related to $(s', w'_1 \cdots w'_n, m')$ if there exists an element i of the indexing type ι, such that $(s', i) \in$ `step` (s, w_1) and $m' = m \cdot$ `to_monoid`(i) and $w_2 \cdots w_n = w'_2 \cdots w'_n$.

We then take the reflexive and transitive closure of `cstep` which we call `der` and we define the set `accepts` which consists of elements w of type $list($`option` $\alpha)$ such that there exists a final state $S \in$ `accept` so that `der` holds between $($`start`$, w, 1)$ and $(S, \epsilon, 1)$. We take the image of this set under a function which removes instances of `none` (i.e the

[1]The code may be found in the file `bRMA.lean`

empty string) in w to yield the language accepted by the `bRMA` under consideration.

We prove the theorem `to_submonoid_bRMA_correct` using our definitions, corresponding to Proposition 1 in Kambites paper; namely that for any M-automaton there exists an N-automaton that accepts the same language and N is a finitely generated monoid. Although the proposition is somewhat trivial, it provides a template for formal proofs involving our construction of M-automata.

3 Dyck Languages

Let $X = \{(,)\}$ be an alphabet consisting of a left parenthesis "(" and a matching right parenthesis ")". Now consider the set of *well-balanced* strings of parentheses, that is; for each left parenthesis appearing in the string there is a matching right parenthesis appearing after it somewhere in the string. This set is the *Dyck language* over one pair of parentheses.

We may generalize Dyck languages to more than one pair of parentheses and there are many ways to formalize Dyck languages, but perhaps the simplest construction that readily generalizes to any set of parentheses[2] is as follows. Let X be an alphabet and let X^{-1} be the set of formal inverses of the elements of X and consider the monoid with presentation

$$D(X) = \langle x \in X \cup X^{-1} \mid x \cdot x^{-1} = 1 \rangle.$$

The identity of this monoid is the empty word ϵ, and multiplication in this monoid is given by concatenation of words followed by reducing the word according to the relation specified in the presentation. The Dyck language is then simply the subset of $(X \cup X^{-1})^*$ whose elements are equal to 1 in $D(X)$.

Dyck languages also arise naturally in certain areas of computer science. A *stack* is a data structure that stores some list of elements, we may *push* an element onto the top of the stack and we may *pop* an element off of the top of the stack. We may describe the actions of pushing and popping on a stack by *partial functions* on the free monoid over some alphabet X.

Recall that a *partial function* is a function which is not defined for all inputs. If our stack stores elements of some set X, then at any point, the contents of the stack is a word in the free monoid X^*. Thus the action of pushing an element onto the top of the stack may be given by the partial function[3] $P_x : X^* \to X^*$ with $P_x(w) = wx$,

[2]There is no restriction that the set be countable.

[3]Note that this function is actually *total*, that is; defined for every input.

and we may describe the action of popping an element from the top of the stack by $Q_x : X^* \to X$ with $Q_x(wx) = w$. We note that Q_x is the right inverse of P_x. If we consider the submonoid of the monoid of partial functions over X^* generated by P_x and Q_x, then we arrive at the *polycyclic monoid* over X, denoted $P(X)$. If we identify x with P_x and x^{-1} with Q_x, then we have the monoid presentation, $P(X) = \langle x \in X \cup X^{-1} \mid x \cdot x^{-1} = 1 \text{ and } x \cdot y^{-1} = 0 \text{ for } x \neq y \rangle$. It is easy to see that the set of words over $X \cup X^{-1}$ equal to 1 in $P(X)$ is precisely the set of words that are equal to 1 in $D(X)$.

So far we have defined what is known as the one-sided or semi-Dyck language, we now turn our attention to the *two-sided Dyck language.* Let $X = \{(,)\}$ be our alphabet, we now remove the restriction that every left parenthesis has a right parenthesis appearing after it somewhere in the string. Instead we simply insist that every left parenthesis has a corresponding right parenthesis somewhere in the string (and vice versa). To construct the *two-sided Dyck language*, we use a group presentation given by

$$F(X) = \langle x \in X \cup X^{-1} \mid x \cdot x^{-1} = 1 = x^{-1} \cdot x \rangle.$$

The set of words over $X \cup X^{-1}$ that are equal to 1 in $F(X)$ is then the *two-sided Dyck language* over X. The group $F(X)$ is known as the *free group* over X and is a well-known and well-studied group that is ubiquitous in combinatorial group theory. It is easy to see that $D(X)$ forms a proper submonoid of $F(X)$.

3.1 A Formal Construction of $D(X)$

In our formal construction of $D(X)$ in LEAN, we required some important ideas from the theory of term rewriting systems.[4]

Definition (TRS)**.** A *term rewriting system* is an ordered pair $(X, \Rightarrow)$ consisting of an alphabet X and a binary relation $\Rightarrow$ defined on X^*. This relation is called a *rewriting relation.* Denote by $\Rightarrow^*$ the reflexive and transitive closure of $\Rightarrow$. We say that $u \in X^*$ *rewrites* to $v \in X^*$ if $u \Rightarrow^* v$ holds.

For use later on, we define an induction principle for the reflexive and transitive closure of a relation (note that this principle was present in the LEAN library).

Definition (RT-closure induction) Let $R \subseteq X \times X$ be a relation and let R' denote its reflexive and transitive closure. Let $y \in X$. Suppose we wish to prove some property P about elements of X given that

[4]The code may be found in the file `dyck.lean`.

for all $x \in X$ we have $xR'y$. The base case is that P holds for y. If for every $a, b \in X$ we have that aRy and $yR'b$ and P holds for b implies P holds for a then P holds for every $x \in X$.

Given our monoid presentation of $D(X)$, we may construct $D(X)$ as the quotient of the free monoid over X by the relation specified. To do this let $u \cdot x \cdot x^{-1} \cdot v \Rightarrow uv$ be the relation, $u \cdot x \cdot x^{-1} \cdot v$ *reduces* to uv in a *single step* where $u, v \in X^*$ and $x, x^{-1} \in X \cup X^{-1}$. Taking the quotient the free monoid X^* by single-step reduction yields the monoid $D(X)$ with identity equal to the empty string and multiplication corresponding to concatenation of words followed by single-step reduction.

We will denote by $\Rightarrow^*$ the reflexive and transitive closure of $\Rightarrow$. We proved that two elements $w, w' \in D(X)$ are equal iff there exists a word u such that $w \Rightarrow^* u$ and $w' \Rightarrow^* u$. This fact allows us to view $D(X)$ as a term rewriting system, and subsequently, simplify many proofs. In proving this fact we proved that $D(X)$ when viewed as a TRS has the Church-Rosser property defined as follows.

Definition (Church-Rosser Property)**.** A term rewriting system $(X, \Rightarrow)$ has the *Church-Rosser property* if for $u, v, v' \in X^*$, we have that $u \Rightarrow^* v$ and $u \Rightarrow^* v'$ implies that there exists a word w such that $v \Rightarrow^* w$ and $v' \Rightarrow w$.

3.2 The Relationship Between $D(X)$ and $F(X)$

We conclude this section by defining the notion of positive and negative elements in $D(X)$ and $F(X)$ and prove some useful lemmas. Elements from X are called *positive generators* and elements from X^{-1} are called *negative generators*. If a word $w \in (X \cup X^{-1})^*$ can be written as a non-empty product of positive generators then it is a *positive element*, negative elements are defined similarly. The following lemmas and theorems will be necessary in the proof of Proposition 14.

We first prove Lemma 12 from Kambites paper. In doing so we were required to prove the following.

Lemma 3 *Let G be a group. Then for all $x, y, z \in G$, $xyz = 1$ iff $yzx = 1$.*

Proof. Let $x, y, z \in G$. Now

$$\begin{aligned} xyz = 1 &\iff yz = x^{-1} \\ &\iff yxz = x^{-1}x = 1. \end{aligned}$$

□

Lemma 4 *If xw represent the identity in $F(X)$, then w has prefix ex^{-1} where e represents the identity.*

Proof. We have that $xw \Rightarrow^* \epsilon$ so that $w \Rightarrow^* x^{-1}$. We now proceed by induction on $\Rightarrow^*$. In the case where $w = x^{-1}$, then it suffices to take $e = \epsilon$. Otherwise suppose $w \Rightarrow w'$ and $w' \Rightarrow^* x^{-1}$. Since $w' \Rightarrow x^{-1}$ there exists words u, v and $y \in X \cup X^{-1}$ such that $w = uyy^{-1}v$ and $w' = uv$. The induction hypothesis tells us that there exists a word e' representing the identity and $e'x^{-1}$ is a prefix of uv. We have to prove that there exists an e that represents the identity such that ex^{-1} is a prefix of $uyy^{-1}v$. Write $ex^{-1}s = uv$. Applying the necessary lemmas we have the following cases

Case: $u = e'u'$ and $x^{-1}s = u'v$ for some u'.

We have two more cases.

Case: $u' = \epsilon$ and $x^{-1}s = v$.

Since $u = e'$ it suffice to take $e = u'yy^{-1}$ which represents the identity and $u'yy^{-1}x^{-1}$ is a prefix of $uyy^{-1}v$ since v begins with x^{-1}.

Case: $u' = x^{-1}c$ and $s = cv$ for some c.

Now we have that $u = e'x^{-1}c$ so it suffices to take $e = e'$.

Case: $e' = uu'$ and $v = u'x^{-1}s$ for some u'.

Here it suffices to take $e = uyy^{-1}u'$ so that this represents $e = uu'$ which represents the identity and $uyy^{-1}u'x^{-1}$ is clearly a prefix of $uyy^{-1}v$ since $v = u'x^{-1}s$.

□

Lemma 5 *If ux represent the identity in $F(X)$, then u has suffix $x^{-1}e$ where e represents the identity.*

Proof. Omitted. □

With the above lemmas we may prove Lemma 12.

Lemma 6 *If uxv represents the identity in $F(X)$, then either u has suffix $x^{-1}e$ or v has prefix ex^{-1} where e represents the identity.*

Proof. By lemma 3 we have that xvu represents the identity and so by lemma 4 we have that there exists a prefix $e'x^{-1}$ of vu where e' represents the identity. Therefore we can write $e'x^{-1}s = vu$. We have the following cases

Case: $v = e'u'$ and $x^{-1}s = u'u$ for some u'.

We have two more cases.

Case: $u' = \epsilon$ and $x^{-1}s = u$.
Now we have $v = e'$ and so we have ux represents the identity so apply lemma 5

Case: $u' = x^{-1}c$ and $s = cu$ for some c.
Here we have $v = e'x^{-1}c$ so set $e = e'$ and v clearly has prefix ex^{-1}.

Case: $e' = vu'$ and $u = u'x^{-1}s$ for some u'.
In this case we have $xvu = xvu'x^{-1}s = xe'x^{-1}s$ which represents the identity so that s represents the identity. Therefore set $e = s$ and so u has suffix $x^{-1}e$.

□

Lemma 7 *If a word $w \in (X \cup X^{-1})^*$ represents the identity $D(X)$ then every prefix of w represents a positive or identity element in $D(X)$.*

Proof. Suppose that w represents the identity. Then we know that w reduces to ϵ when we view $D(X)$ as a TRS. We proceed by induction on the rewrite relation. In the case where $w = \epsilon$, we have that p is a prefix of ϵ so $p = \epsilon$ and p represents the identity in $D(X)$. Now let $w, w' \in (X \cup X^{-1})^*$ and suppose that $w \Rightarrow w'$ and $w' \Rightarrow^* \epsilon$. Since $w \Rightarrow w'$, we know that there exists $u, v \in (X \cup X^{-1})$ and $x \in X$ such that $w = uxx^{-1}v$ and $w' = uv$. The induction hypothesis tells us that every prefix of uv is a positive element or represents the identity.

Let p be a prefix of $uxx^{-1}v$. Then we have that there exists some word s such that $ps = uxx^{-1}v$. Applying the necessary lemmas regarding equations in the free monoid we arrive at the following case distinctions:

Case: $u = pu'$ and $s = u'xx^{-1}v$ for some u'.
In the first case we have that p is a prefix of u so the induction hypothesis tells us that p is a positive or identity element.

Case: $p = uu'$ and $xx^{-1}v = u's$ for some u'.
We then have another two cases.

Case: $u' = \epsilon$ and $s = xx^{-1}v$
In this case $p = u$ and so p is a prefix of u. The induction hypothesis tells us that p is a positive or identity element.

Case: $u' = xc$ and $x^{-1}v = cs$ for some c.
We have another two cases.

Case: $c = \epsilon$ so that $u' = x$ and $s = x^{-1}v$.
Now u is a positive or identity element, in either case $p = uu' = ux$ is positive.

Case: $c = x^{-1}d$ and $v = ds$.

In this case we have that $p = uxx^{-1}d$. Now the induction hypothesis tells us that ud is positive or represents the identity (since $v = ds$ it follows that ud is a prefix of $uds = uv$). In either case we have that $uxx^{-1}d$ represents the element ud so it follows $p = uxx^{-1}d$ represents a positive or identity element.

□

Lemma 8 *If a word $w \in (X \cup X^{-1})^*$ represents the identity $D(X)$ then every suffix of w represents a negative or identity element in $D(X)$.*

Proof. Omitted. □

The above two lemmas lend themselves to sufficient and necessary conditions for a word to represent the identity in $D(X)$.

Theorem 1 *Every prefix of w represents a positive or identity element in $D(X)$ and every suffix of w represents a negative or identity element in $D(X)$ iff w represents the identity in $D(X)$.*

Proof. ($\Rightarrow$) Firstly we have that w represents a positive or identity element and w represents a negative or identity element as w is a prefix/suffix of itself. We can not have that w be both positive and negative, and similarly we can not have that w be both the identity and positive/negative, so it must be the case that w represents the identity in $D(X)$. ($\Leftarrow$) Follows from lemma 7 and lemma 8. □

Lemma 9 *Let $w \in (X \cup X^{-1})^*$, then every prefix of w represents a positive or identity element in $D(X)$ iff every prefix of w represents a positive or identity element in $F(X)$.*

Proof. It is clear that any negative element in $D(X)$ is a negative element in $F(X)$ and any word representing the identity in $D(X)$ also represents the identity in $F(X)$.

We now proceed by induction on the length of the word w. In the case where $w = \epsilon$, we have that w every prefix of w (being just ϵ) represents the identity in $D(X)$. If $w = x$ for some $x \in (X \cup X^{-1})$, then the only prefix of w is x and we cannot have that x represents the identity in $F(X)$, so it must be positive in $F(X)$. Now x is a single positive generator and hence is positive in $D(X)$.

Assume now as our induction hypothesis that, if every prefix of $w = w_1 \cdots w_n$ represents a positive or identity element in $F(X)$, then every prefix of w represents a positive or identity element in $D(X)$.

Now suppose every prefix of $w \cdot w_{n+1}$ represents a positive or identity element in $F(X)$. The induction hypothesis tells us that every prefix of w represents a positive or identity element in $D(X)$. Now let p be an arbitrary prefix of $w \cdot w_{n+1}$. This implies that either p is a prefix of w in which case p represents a positive or identity element in $D(X)$ or $p = w \cdot w_{n+1}$. We know that w represents a positive or identity element in $D(X)$. We also know that p represents a positive or identity element in $F(X)$. Therefore we have four cases.

Case: p is positive in $F(X)$ and w is positive in $D(X)$

In this case since $p = w \cdot w_{n+1}$ is positive, we have to check the two cases corresponding to the case that w_{n+1} is a positive generator say x or a negative generator x^{-1}. In the case that $w_{n+1} = x$, we have that wx is positive in $D(X)$. In the case where $w_{n+1} = x^{-1}$, we know that since w is positive in $D(X)$ we have that w can be written as a product w' of positive generators so that $w'x^{-1}$ is positive. This implies $w' = ux$ for some u. Now p represents uxx^{-1} which represents u in $D(X)$. If $u = \epsilon$, then u represents the identity in $D(X)$, otherwise since $w' = ux$ and u is non-empty, it follows u is positive in $D(X)$.

Case: p is positive in $F(X)$ and w represents the identity in $D(X)$.

In this case $p = w \cdot w_{n+1}$ represents w_{n+1} in $F(X)$ and since this is positive we must have that w_{n+1} is a positive generator and hence positive in $D(X)$.

Case: p represents the identity in $F(X)$ and w is positive in $D(X)$

Suppose $w_{n+1} = x$. Since w is positive and wx represents the identity, we must have that w represents x^{-1} in $F(X)$. This contradicts the fact that w is positive in $D(X)$, so we must have $w_{n+1} = x^{-1}$. In this case w represents x in $F(X)$ and $D(X)$ so that $p = wx^{-1}$ represents xx^{-1} which represents the identity in $D(X)$.

Case: p represents the identity in $F(X)$ and w represents the identity in $D(X)$

Here we have a contradiction as $p = w \cdot w_{n+1}$ represents w_{n+1} in $F(X)$ which cannot possibly be the identity. Thus this case is not possible.

□

Lemma 10 *Let $w \in (X \cup X^{-1})^*$, then every suffix of w represents a negative or identity element in $D(X)$ iff every suffix of w represents a negative or identity element in $F(X)$.*

Proof. Omitted. □

4 The Representation Theorem

In light of the connections between the theory of rational transductions and M-automata, the Chomsky-Schutzenberger representation theorem has the following interpretation in the M-automata setting

(i) The language L is context-free.

(ii) The language L is accepted by a polycyclic monoid M-automaton.

(iii) The language L is accepted by a free group M-automaton.

Kambites proof is that (*ii*) implies (*iii*), that is; given a language accepted by a polycyclic monoid M-automaton, we can construct an M-automaton over some free group that accepts the same language.

Denote by $X^{\#}$ to be the set $(X \cup \{\#\}) \cup (X \cup \{\#\})^{-1}$ where $\#$ is a symbol not in X. The main step in the proof involves padding words in $(X \cup X^{-1})$ with a new symbol $\#$ in such a way so that a word represents the identity in $P(X)$ iff it admits a padding that represents the identity in $F(X^{\#})$. These padded words may be then used to construct an $F(X^{\#})$ automaton accepting the same language as a $P(X)$ automaton. As we will be proving statements about the set of words representing the identity in $P(X)$, it suffices that these statements hold for elements representing the identity in $D(X)$.

We recall Kambites definition of an element x in the free group being a *minimum* of a word w in the free group.

Definition We say that $x \in F(X)$ is a *minimum* of w if w has a prefix representing x in $F(X)$ and no prefix which represents x is immediately followed by a negative generator.

4.1 Permissible Paddings and Regular Languages

Kambites defines a *permissible padding* of a word $w_1 \cdots w_n \in X^*$ to be a word of the form $x_1 \cdots x_n (\#^{-1})^k$ where $k \in \mathbb{N}$ and each x_i is of the form $w_i\#$ if w_i is a positive generator and $(\#^{-1})^k w_i \#$ where $k \in \mathbb{N}$ if w_i is a negative generator.

In order to formally define permissible paddings, we begin with a brief discussion about regular languages and regular expressions. Let X be an alphabet, a *regular expression* is defined inductively as follows:

(i) 0 and 1 are regular expressions.

(ii) If $x \in X$, then x is a regular expression.

(iii) If x and y are regular expressions then so is their product $x \cdot y$, and their sum $x + y$.

(iv) If x is a regular expression, then x^* is a regular expression.

Let the set of all regular expressions over X be denoted $Reg(X)$. We then endow regular expressions with the following operation which when applied to a regular expression, generates a language known as a *regular language*, this operation is defined recursively. Let $\|\cdot\| : Reg(X) \to \mathcal{P}(X^*)$ be defined by,

$$\begin{aligned}
\|0\| &= \emptyset \\
\|1\| &= \{\epsilon\} \\
\|x\| &= \{x\} \text{ where } x \in X \\
\|x+y\| &= \|x\| \cup \|y\| \\
\|x \cdot y\| &= \|x\| \cdot \|y\| \\
\|x^*\| &= (\|x\|)^*
\end{aligned}$$

where the product of two sets $A, B \subseteq X^*$ is defined as the set containing all words whose prefixes come from A and whose suffixes come from B (i.e the elements of A “followed” by the elements of B). The star of a set $A \subseteq X^*$ is defined as all words that can be formed as a finite concatenation of elements of A. As an example consider $(\{ab\})^*$ where $a, b \in X$, then this the set consisting of the elements $\{\epsilon, ab, abab, ababab, ...\}$. So $(ab)^*$ is simply ab repeated zero or more times.

Remark Regular languages are connected to our definition of M-automata by letting the monoid $M = \{1\}$. The resulting automaton is equivalent to a non-deterministic finite state machine, which recognizes regular languages.

Given a word $w \in X^*$, we may use a regular expression to define the set of permissible paddings[5] of w. Define the function $f : X \cup X^{-1} \to Reg(X)$ to pad a single letter as follows,

$$f(x) = \begin{cases} x \cdot \# & \text{if } x \text{ is a positive generator} \\ (\#^{-1})^* \cdot x \cdot \# & \text{if } x \text{ is a negative generator} \end{cases}$$

Then for $x_1 \cdots x_n$ define the function $pad : (X \cup X^{-1})^* \to Reg(X^{\#})$ by setting

$$pad(x_1 \cdots x_n) = \left(\prod_{i=1}^{n} f(x_i) \right) \cdot (\#^{-1})^*$$

The set of permissible paddings of a word w is then simply $\|pad(w)\|$. We call a padding of a letter $x \in X \cup X^{-1}$ (i.e an element of the language generated by $f(x)$) to be a *letter padding* and we call a padding of the form $\prod_{i=1}^{n} f(x_i)$ a *semi-padding.* We conclude this section with some useful lemmas about permissible paddings.

[5]May be found in the file `permissible_paddings.lean`

Lemma 11 *Every semi-padding of w is a permissible padding of w.*

Proof. A permissible padding of w consists of a semi-padding followed by the symbol $\#^{-1}$ zero or more times. Thus every semi-padding w' of w is also a permissible padding with $\#^{-1}$ occurring zero times after w'. $\square$

Lemma 12 *Semi-paddings are closed under multiplication in the sense that if u' is a semi-padding of u and v' is a semi-padding of v, then $u'v'$ is a semi-padding of uv.*

Proof. Let $u = u_1 \cdots u_n$ and $v = v_1 \cdots v_m$. Suppose u' is a semi-padding of u and v' is a semi-padding of v. We have to show that $u'v' \in \| \prod_{i=1}^{n} f(u_i) \cdot \prod_{i=1}^{m} f(v_i)\|$ equivalently we need to show $u' \in \| \prod_{i=1}^{n} f(u_i)\|$ and $v' \in \| \prod_{i=1}^{m} f(v_i)\|$ by definition of the product of two sets in the free monoid. Now this follows from the fact that u' is a semi-padding of u and v' is a semi-padding of v. $\square$

Lemma 13 *If u' is a semi-padding of u and v' is a permissible padding of v, then $u'v'$ is a permissible padding of uv.*

Proof. Since v' is a permissible padding of v, it is the product of a semi-padding q of v and the symbol $\#^{-1}$ following q some number of times. Let $q' = (\#^{-1})^k$ for some $k \in \mathbb{N}$ and write $v' = qq'$. Since semi-paddings are closed under multiplication we have that $u'q$ is a semi-padding of uv so that $u'v' = u'qq'$ is a permissible padding of uv. $\square$

Lemma 14 *If $u\#^{-1}v$ is a semi-padding of w, then $u\#^{-1}\#^{-1}v$ a semi-padding of w.*

Proof. Since $u\#^{-1}v$ is a semi-padding of w, we can split the word $u\#^{-1}v$ into a product of letter paddings where the given occurrence of $\#^{-1}$ occurs in one of these letter paddings. This letter padding must pad a negative generator since $\#^{-1}$ cannot appear as a padding of a positive generator. By the definition of a letter padding, we can repeat $\#^{-1}$ any number of times before the generator, so we can adjoin $\#^{-1}$ on the left of this generator and the resulting word is still a semi-padding of w. $\square$

Lemma 15 *If $u\#^{-1}v$ is a permissible padding of w, then the word formed by deleting all occurrences of $\#$ and $\#^{-1}$ from v must either be the empty word ϵ or must begin with a negative generator.*

Proof. Let v' denote the word formed by deleting all occurrences of # and $\#^{-1}$ from v. Suppose $u\#^{-1}v$ is a permissible padding of w. It must be the case that the given occurrence of $\#^{-1}$ either pads the end of w or occurs in a letter padding of a negative generator. In the first case we have that v' is the empty word and in the second case we have that v' begins with some negative generator as $\#^{-1}$ may only appear before a negative generator. □

Remark. Type theoretically, we use the sum type $\oplus$ to adjoin the single point type `unit` (which we choose to mean #) to our alphabet α to get $\alpha \oplus$ `unit` as a new type whose elements consist of either elements of type α or the single element of the `unit` type. This is the type theoretical analogue of the disjoint union of two sets.

4.2 Proposition 14

With all of these definitions and lemmas at hand we may begin to prove Proposition 14 which states that the following are equivalent if w is a word representing the identity in $F(X)$

(i) w represents the identity in $D(X)$

(ii) Every prefix of w represents a positive or identity element in $F(X)$

(iii) The only minimum of w is the identity of $F(X)$.

Note: We were not able to formally verify that (*iii*) implies (*ii*).

Theorem 2 (*i*) $\Rightarrow$ (*ii*) *Suppose w represents the identity in $D(X)$, then every prefix of w represents a positive or identity element in $F(X)$.*

Proof. Since w represents the identity in $D(X)$, we have that every prefix of w represents a positive or identity element in $D(X)$ and hence $F(X)$ by lemma 9. □

To prove that (*ii*) implies (*i*) we require the following,

Lemma 16 *If w represents the identity in $F(X)$ and every prefix of w represents a positive or identity element, then every suffix represents a negative or identity element.*

Proof. Since w represents the identity in $F(X)$, we have that $w \Rightarrow^* \epsilon$. We proceed by induction on the rewrite relation of $F(X)$ when viewed as a TRS. In the case where $w = \epsilon$, we have that every prefix/suffix is simply ϵ which clearly represents the identity in $F(X)$ so that every suffix represents the identity.

Otherwise suppose that $w \Rightarrow w'$ and $w' \Rightarrow^* \epsilon$. It follows that there exists words u, v and $x \in X$ such that $uxx^{-1}v \Rightarrow uv$ and $uv \Rightarrow^* \epsilon$. The induction hypothesis tells us that if every prefix of uv represents a positive or identity element then every suffix of uv represents a negative or identity element.

Suppose that every prefix of $uxx^{-1}v$ represents a positive or identity element. It is easy to see that every prefix of uv also represents a positive or identity element so that we have that every suffix of uv represents a negative or identity element.

Now let s be an arbitrary suffix of $uxx^{-1}v$ and write $uxx^{-1}v = ts$. Making repeated use of the necessary lemmas about equations in the free monoid, we identify the following possible cases:

Case: $u = tu'$ and $s = u'xx^{-1}v$ for some u'.
In this case we have that $u'v$ is a suffix of uv so that $u'v$ represents a negative or identity element. In the case that $u'v$ is a negative element it follows that since s reduces to $u'v$ that s is negative element. If $u'v$ represents the identity then s represents the identity for the same reason.

Case: $t = uu'$ and $xx^{-1}v = u's$ for some u'
Here we have two more cases.

- Case: $u' = \epsilon$ and $xx^{-1}v = s$.
  In this case we have that v is a suffix of uv so represents a negative or identity element, in which case it follows that since $xx^{-1}v = s$ reduces to v that s represents a negative or identity element.
- Case: $u' = xc$ and $x^{-1}v = cs$ for some c.
  We have two more cases.
  - Case: $c = \epsilon$ and $x^{-1}v = s$.
    In this case we have v is negative or represents the identity. If v is negative then so is $s = x^{-1}v$ and if v represents the identity then s reduces to x^{-1} which is negative.
  - Case: $c = x^{-1}d$ and $v = ds$ for some d.
    Since s is a suffix of v which is a suffix of uv it follows that s is negative or represents the identity.

□

Theorem 3 $(ii) \Rightarrow (i)$ *If every prefix of w represents a positive or identity element in $F(X)$ and w represents the identity in $F(X)$ then w represents the identity in $D(X)$.*

Proof. We have that every prefix of w represents a positive or identity element in $F(X)$ and hence $D(X)$ by lemma 9. By the previous lemma, every suffix of w represents a negative or identity element in $F(X)$ and hence $D(X)$. Then by Theorem 1 w represents the identity in $D(X)$. $\square$

Theorem 4 *(ii) $\Rightarrow$ (iii) Let w be a word representing the identity in $F(X)$ and suppose that every prefix of w is a positive or identity element in $F(X)$, then the only minimum of w is the identity of $F(X)$.*

Proof. First we show that the identity is indeed a minimum of w. Since w represents the identity, it has a prefix representing the identity, namely w. Now let p and s be any words such that $w = ps$. Suppose that p represents the identity. We must show that $s = \epsilon$ or $s = xs'$ for some s'. In the case where $w = p$, we have that $s = \epsilon$. Now assume s is non-empty. We can then write $s = xs'$ where $x \in (X \cup X^{-1})$. Now we have $w = pxs'$ and so px represents a positive or identity element. In the case where px is positive we have that since p represents the identity we have that px represents x which is positive so that x is a positive generator. Otherwise if px represents the identity we would have that x represents the identity which is not true so this case is not possible.

Now we prove that the identity is the only minimum of w. Let u be a non-identity minimum of w. Then w has a prefix p representing u and every prefix of w representing u is not immediately followed by a negative generator. Write $w = ps$ for some s. By lemma 16 we have that every suffix of w represents a negative or identity element so that s represents a negative or identity element. We also have p represents a positive or identity element. Clearly p cannot represent the identity as this would contradict the fact that u does not represent the identity. Thus we have two cases.

Case: p is positive and s is negative.

In this case since s is negative it has some prefix ex^{-1} for some $x \in X$ where e represents the identity. Therefore write $w = pex^{-1}s'$ for some s'. Now since u is a minimum of w and pe is a prefix representing u, this prefix cannot be followed by a negative generator contradicting the fact that pe is followed by x^{-1}.

Case: p is positive and s represents the identity.

In this case we have that $w = ps$ represents p which must represent the identity since w represents the identity contradicting the fact that p is positive.

$\square$

4.3 Lemma 15

We were not able to verify Lemma 15 as is, but we provide a sufficient condition in the code for Lemma 15 to hold. To make up for this, we provide an informal proof of the sufficient condition.[6]

Lemma 17 *Let w be a word representing the identity in $D(X)$ and suppose $w = uxv$ where $u, v \in (X \cup X^{-1})^*$ and $x \in X$. Then v has prefix ex^{-1} where e represents the identity in $D(X)$.*

Proof. We must have that the reduction process sends each x to some given occurrence of x^{-1} by deleting the letters between them. This given occurrence must appear after x since $x^{-1}x$ is irreducible in $D(X)$. The product of the letters deleted must be some factor representing the identity, so set e equal to this product and the claim follows. $\square$

We are now ready to prove Lemma 15.

Lemma 18 *Let uv be a word representing the identity in $D(X^\#)$. Then there exists $s, t \in X^\#$ such that $v = ts$ and*

(i) $t = \epsilon$ *or* t *begins with a negative generator; and*

(ii) $u\#s\#^{-1}t$ *also represents the identity in* $D(X^\#)$.

Proof. Since uv represents the identity in $D(X^\#)$, we have that every prefix of uv represents a positive or identity element in $D(X^\#)$. Since u is a prefix of uv, it follows that u is positive or represents the identity. If u represents the identity, then v represents the identity and the claim follows by setting $s = v$ and $t = \epsilon$. Otherwise u is positive and hence we can write u as a product u' of positive generators, so that u' and u are equal in $D(X^\#)$. It follows we can write $u' = wx$ and since wxv represents the identity, we may use lemma 17 to write $v = ex^{-1}v'$ where e represents the identity. Now set $s = e$ and $t = x^{-1}v'$, so we have $wx\#e\#^{-1}x^{-1}v'$ represents the identity as required. $\square$

4.4 Proposition 16

We have that the following are equivalent,

(i) The word w represents the identity in $D(X)$.

(ii) The word w admits a permissible padding that represents the identity in $D(X^\#)$.

(iii) The word w admits a permissible padding that represents the identity in $F(X^\#)$.

[6] The proof of Lemma 15 is formally verified and present in the code, it simply rests on an assumption for which we have no formal proof.

Theorem 5 *$(i) \Rightarrow (ii)$ Suppose w represents the identity in $D(X)$, then w admits a permissible padding which represents the identity in $D(X^{\#})$.*

Proof. We proceed by induction on the rewrite relation. In the case where $w = \epsilon$, then ϵ is a permissible padding of itself and clearly represents the identity in $D(X^{\#})$.

Now suppose that $w \Rightarrow w'$ and $w' \Rightarrow^* \epsilon$. Then there exists words u, v and a letter $x \in X$ such that $w = uxx^{-1}v$ and $w' = uv$. The induction hypothesis tells us that uv admits a permissible padding representing the identity in $D(X^{\#})$. Therefore there exists a semi-padding u' of u and a permissible padding v' of v such that $u'v'$ represents the identity in $D(X^{\#})$. It then follows by lemma 18 that there exists s, t such that $v' = st$ and $t = \epsilon$ or t begins with a negative generator and $u'\#s\#^{-1}t$ also represents that identity in $D(X^{\#})$. If $t = \epsilon$ then we have $v' = s$ and $u'\#s\#$ represents the identity. We may pad $uxx^{-1}v$ as

$$u'x\#\#^{-1}x^{-1}\#s\#^{-1}.$$

We have that u' is a semi-padding of u and $x\#\#^{-1}x^{-1}\#$ is a semi-padding of xx^{-1} and so $u'x\#\#^{-1}x\#$ is a semi-padding of uxx^{-1} so that $u'x\#\#^{-1}x\#s$ is a permissible padding of $uxx^{-1}v$ and we may append $\#^{-1}$ on the end of a permissible padding as many times as we want and the result is still a permissible padding of $uxx^{-1}v$ so that $u'x\#\#^{-1}x^{-1}\#s\#^{-1}$ is a permissible padding of $uxx^{-1}v$ representing the identity. To see this note that $u'x\#\#^{-1}x^{-1}\#s\#^{-1}$ reduces to $u'\#s\#^{-1}$ which we know represents the identity.

Otherwise if t begins with a negative generator then we have that $u\#s\#^{-1}t$ represents the identity. We may pad $uxx^{-1}v$ as

$$u'x\#\#^{-1}x^{-1}\#s\#^{-1}t.$$

Since t begins with a negative generator we have that $s\#^{-1}t$ is also a permissible padding of v so that $u'x\#\#^{-1}x^{-1}\#s\#^{-1}t$ is a permissible padding representing the identity in $D(X^{\#})$. □

Note: The proof of $(ii) \Rightarrow (iii)$ and $(iii) \Rightarrow (i)$ was formalized exactly as Kambites wrote it and so there is no need to repeat it here.

Concluding Remarks We would like to note that the vast majority of all statements formalized had a constructive proof (which therefore has interesting computational interpretations as programs). As the formalization of M-automata was done early on in this project, there is much room for redesign.

In particular it seems that it would be more useful for LEAN if our construction M-automata were designed in such a way that would allow for computation. That is, as opposed to using a relation, we would write a program that executes the steps that an M-automaton would take. This way, we would not have to prove the decidability of membership of the language accepted by a given M-automaton (when the underlying monoid type has decidable equality). Instead we would get decidability of membership for “free” so to speak.

Clearly more work can be done in formally verifying Lemma 15 and implication $(iii) \Rightarrow (ii)$ of Proposition 14. One thing we would like to note is that the restriction to polycyclic monoids/free groups of countable rank may be unnecessary. This restriction was not used in the formalization, however until every statement is fully formalized we cannot be certain of this. Indeed the theorem may hold for polycyclic monoids/free groups of uncountable rank.

5 References

[1] Kambites, M., 2009. Formal Languages and Groups as Memory. Communications in Algebra, 37(1), pp.193-208.
[2] Greibach, S., 1978. Remarks on blind and partially blind one-way multicounter machines. Theoretical Computer Science, 7(3), pp.311-324.
[3] Leanprover-community.github.io. 2022. Mathematics in mathlib. [online] Available at: `<https://leanprover-community.github.io/mathlib-overview.html>` [Accessed 13 May 2022].
[4] Lawson, M., 1999. Inverse semigroups. Singapore: World Scientific, p.285.