Chapter 2

Preliminary Definitions

We now review some notions that will be used in this thesis, as well as the main definition of shuffle on trajectories, which will be used throughout this thesis. Readers familiar with the concepts below should feel free to consult this chapter only as necessary.

2.1 Formal Language Theory

For additional background in formal languages and automata theory, please see Yu [201] or Hopcroft and Ullman [68]. Let Σ be a finite set of symbols, called *letters*. The set Σ is an *alphabet*. Then Σ^* is the set of all finite sequences of letters from Σ, which are called *words*. The empty word ϵ is the empty sequence of letters. Given two words $w = w_1 w_2 \cdots w_n$ and $x = x_1 \cdots x_m$ where $x_i, w_j \in \Sigma$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$, their concatenation wx is the word $w_1 w_2 \cdots w_n x_1 x_2 \cdots x_m$. The *length* of a word $w = w_1 w_2 \cdots w_n \in \Sigma^*$, where $w_i \in \Sigma$, is n, and is denoted $|w|$. Note that ϵ is the unique word of length 0. A *language* L is any subset of Σ^*. If $w \in \Sigma^*$, we will denote the language consisting only of w by w instead of $\{w\}$. For a language $L \subseteq \Sigma^*$, by $|L|$ we denote its cardinality as a set.

Let $L \subseteq \Sigma^*$ be a language. By $\overline{L}$, we mean $\Sigma^* - L$, the complement of L. Let L_1, L_2 be languages. By $L_1 L_2$ we mean the concatenation of L_1 and L_2, given by $L_1 L_2 = \{xy \ : \ x \in L_1, y \in$

$L_2\}$. If L is a language and $n \geq 0$, then the set L^n is defined recursively as follows: $L^0 = \{\epsilon\}$, $L^{n+1} = L^n L$ for all $n \geq 0$. We denote $L^* = \cup_{n\geq 0} L^n$ and $L^+ = \cup_{n\geq 1} L^n$. If $L_1, \ldots, L_k \subseteq \Sigma^*$ are languages, we use the notation $\prod_{i=1}^k L_i = L_1 L_2 \cdots L_k$. If L is a language and k is a natural number, then we denote $L^{\leq k} = \cup_{i=0}^k L^i$.

Given two languages $L_1, L_2 \subseteq \Sigma^*$, the *left quotient* of L_2 by L_1 is denoted $L_1 \setminus L_2$ and is given by

$$L_1 \setminus L_2 = \{x \in \Sigma^* \; : \; \exists y \in L_1 \text{ such that } yx \in L_2\}.$$

Similarly, the *right quotient* (or simply *quotient*, if there is no confusion) of L_2 by L_1 is denoted L_2/L_1 and is given by

$$L_2/L_1 = \{x \in \Sigma^* \; : \; \exists y \in L_1 \text{ such that } xy \in L_2\}.$$

The *shuffle* operation is defined as follows: if $x, y \in \Sigma^*$ are words, then the shuffle of x and y, denoted $x \sqcup\!\sqcup\, y$ is defined by

$$x \sqcup\!\sqcup\, y = \{\prod_{i=1}^n x_i y_i \; : \; x = \prod_{i=1}^n x_i, \; y = \prod_{i=1}^n y_i; \; x_i, y_i \in \Sigma^* \; \forall 1 \leq i \leq n\}.$$

If L_1, L_2 are languages, then $L_1 \sqcup\!\sqcup\, L_2$ is given by $L_1 \sqcup\!\sqcup\, L_2 = \{x \sqcup\!\sqcup\, y \; : \; x \in L_1, y \in L_2\}$.

We denote by $\mathbb{N}$ the set of natural numbers: $\mathbb{N} = \{0, 1, 2, \ldots\}$. If we wish to refer to the positive numbers, we will use the notation $\mathbb{N}^+ = \{1, 2, \ldots, \}$. Let $I \subseteq \mathbb{N}$. If there exist $n_0, p \in \mathbb{N}$, $p > 0$, such that for all $x \geq n_0$, $x \in I \iff x + p \in I$, then we say that I is *ultimately periodic* (u.p.). For $n, m \in \mathbb{N}$, we use the notation $m|n$ to denote that m is a divisor of n, that is, there exists $k \in \mathbb{N}$ such that $n = km$.

Given a set X, we use the notation $2^X = \{Y \; : \; Y \subseteq X\}$. Given alphabets Σ, Δ, a *morphism* is a function $h : \Sigma^* \to \Delta^*$ satisfying $h(xy) = h(x)h(y)$ for all $x, y \in \Sigma^*$. Given a morphism $h : \Sigma^* \to \Delta^*$ and a language $L \subseteq \Sigma^*$, then the image of L under h is given by $h(L) = \{h(x) \; : \; x \in L\}$, while if $L' \subseteq \Delta^*$, the inverse image of L' under h is defined by $h^{-1}(L') = \{x \in \Sigma^* \; : \; h(x) \in L'\}$. A *substitution* is a function $h : \Sigma^* \to 2^{\Delta^*}$ satisfying $h(xy) = h(x)h(y)$ for all $x, y \in \Sigma^*$. Given a substitution $h : \Sigma^* \to 2^{\Delta^*}$ and a language $L \subseteq \Sigma^*$, then the image of L under h is given by $h(L) = \cup_{x \in L} h(x)$. We say that a substitution is *regular* if $h(a) \in$ REG for all $a \in \Sigma$ (see Section 2.2 below for the definitions of the regular languages and REG).

Given a word $w \in \Sigma^*$ and $a \in \Sigma$, $|w|_a$ is the number of occurrences of a in w. For instance, if $w = abbaa$, then $|w|_a = 3$ and $|w|_b = 2$. If $w \in \Sigma^*$ is a word, $\text{alph}(w) = \{a \in \Sigma \ : \ |w|_a > 0\}$. If $L \subseteq \Sigma^*$, $\text{alph}(L) = \cup_{w \in L} \text{alph}(w)$.

For an alphabet $\Sigma = \{a_1, a_2, \ldots, a_n\}$ with a specified order $a_1 < a_2 < \cdots < a_n$, the *Parikh mapping* is given by $\Psi : \Sigma^* \to \mathbb{N}^n$, as follows:

$$\Psi(w) = (|w|_{a_i})_{i=1}^n.$$

It is extended to $\Psi : 2^{\Sigma^*} \to 2^{\mathbb{N}^n}$ as expected. For instance, if $\Sigma = \{a, b\}$ with $a < b$, and $x = abbaa$, then $\Psi(x) = (2, 3)$. If $L = \{a^n b^n a^n \ : \ n \geq 0\}$, then $\Psi(L) = \{(2n, n) \ : \ n \geq 0\}$. The inverse mapping is given by $\Psi^{-1} \ : \ 2^{\mathbb{N}^n} \to 2^{\Sigma^*}$ is given by $\Psi^{-1}(S) = \{u \in \Sigma^* \ : \ \Psi(u) \in S\}$ for all $S \subseteq \mathbb{N}^n$. A language $L \subseteq \Sigma^*$ is said to be *commutative* if $L = \Psi^{-1}(\Psi(L))$. Thus, L is commutative if rearranging the letters from any word in L always yields a word in L. For instance, the language $L = \{aab, aba, baa, ab, ba\}$ is commutative. For any language L, $com(L)$ is the commutative closure of L, i.e., $com(L) = \{v \in \Sigma^* \ : \ \exists u \in L \text{ such that } \Psi(u) = \Psi(v)\}$. For instance, $com(\{abc\}) = \{abc, acb, bac, bca, cab, cba\}$.

We say that a language $L \subseteq \Sigma^*$ is *bounded* if there exist $w_1, w_2, \ldots, w_k \in \Sigma^*$ such that $L \subseteq w_1^* w_2^* \cdots w_k^*$. If L is not bounded we say that it is *unbounded*. The languages $L_1 = \{a^n b^{2n} c^n \ : \ n \geq 0\}$ and $L_2 = (ab)^* + (cd)^*$ are bounded, as $L_1 \subseteq a^* b^* c^*$ and $L_2 \subseteq (ab)^*(cd)^*$. The language $L_3 = \{a, b\}^*$ is known to be unbounded.

2.2 Regular Languages

We now describe finite automata and regular languages. A *deterministic finite automaton* (DFA) is a five-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is an alphabet, $\delta : Q \times \Sigma \to Q$ is a transition function, $q_0 \in Q$ is a distinguished start state, and $F \subseteq Q$ is the set of final states. We extend δ to $Q \times \Sigma^*$ in the usual way: if $q \in Q$ and $w \in \Sigma^*$, then define $\delta(q, w) = q$ if $w = \epsilon$ and

$$\delta(q, w) = \delta(\delta(q, w'), a)$$

if $w = w'a$ for some $w' \in \Sigma^*$ and $a \in \Sigma$.

A word $w \in \Sigma^*$ is *accepted by* M if $\delta(q_0, w) \in F$. The *language accepted* by M, denoted $L(M)$, is the set of all words accepted by M. A language is called *regular* if it is accepted by some DFA.

A *nondeterministic finite automaton* (NFA) is a five-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where Q, Σ, q_0 and F are as in the deterministic case, while $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$ is the nondeterministic transition function. Again, δ is extended to $Q \times \Sigma^*$ in the natural way. To define the action of δ formally, we require a few notions. First define a binary relation $R_\epsilon \subseteq Q^2$. The relation is given by $q_i R_\epsilon q_j$ if $q_j \in \delta(q_i, \epsilon)$. Let R_ϵ^* be the reflexive, transitive closure of R_ϵ. Define $cl : Q \to 2^Q$ by

$$cl(q) = \{q' \ : q R_\epsilon^* q'\}.$$

Thus, $cl(q)$ is the set of all states that are reachable from q by following some path of ϵ-transitions in M. Further, let $cl(S) = \cup_{q \in S} cl(q)$ for all $S \subseteq Q$. We may now define δ as a function from $Q \times \Sigma^*$ to 2^Q: if $q \in Q$ then $\delta(q, \epsilon) = cl(q)$ and for all $a \in \Sigma$ and $w \in \Sigma^*$,

$$\delta(q, wa) = cl\left(\bigcup_{q' \in \delta(q, w)} \delta(q', a)\right).$$

A word w is accepted by M if $\delta(q_0, w) \cap F \neq \emptyset$. It is known that the language accepted by an NFA is regular. We denote the class of regular languages by REG.

For a DFA or NFA M, we say that M is *complete* if δ is a complete function, i.e., if $\delta(q, a)$ is defined for all $q \in Q$ and $a \in \Sigma$.

We can draw a DFA or NFA as a directed graph using the following conventions:

(a) states are drawn as vertices, labelled with their name;

(b) transitions are drawn as directed edges, labelled with the letter of the transition. Thus, if $\delta(q_1, a) = q_2$, there is a directed edge (q_1, q_2) with label a;

(c) final states are indicated as vertices with double circles;

(d) the start state is indicated with an unlabelled arrow entering it.

For example, the DFA given in Figure 2.1 has start state 1, final state set $\{2\}$ and transitions $\delta(1, b) = \delta(2, b) = 1$ and $\delta(1, a) = \delta(2, a) = 2$.

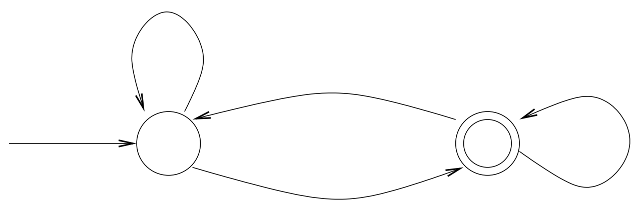

Figure 2.1: A DFA, illustrated.

We also introduce the *Myhill-Nerode congruence* on Σ^*. Given a language $L \subseteq \Sigma^*$, we denote the Myhill-Nerode congruence with respect to L on Σ^* by $\equiv_L$. Given $x, y \in \Sigma^*$, $x \equiv_L y$ if and only if, for all $z \in \Sigma^*$,

$$xz \in L \iff yz \in L.$$

We note that $\equiv_L$ is an equivalence relation and that a language L is regular if and only if $\equiv_L$ has finite index [68, Thm. 3.9].

Finally, we define regular expressions. Let Σ be an alphabet. A *regular expression* is a word over the alphabet $\{\emptyset, \epsilon, (,), *, +\} \cup \Sigma$ defined as follows:

(a) the following are regular expressions: ϵ, $\emptyset$ and a for all $a \in \Sigma$;

(b) if r_1, r_2 are regular expressions, so are $(r_1 r_2)$ and $(r_1 + r_2)$;

(c) if r_1 is a regular expression, so is (r_1^*).

Given a regular expression r, it defines a language $L(r)$ as follows:

(a) $L(\epsilon) = \{\epsilon\}$, $L(\emptyset) = \emptyset$ and $L(a) = \{a\}$;

(b) $L((r_1 + r_2)) = L(r_1) \cup L_{(}r_2)$;

(c) $L((r_1 r_2)) = L(r_1)L(r_2)$;

(d) $L((r_1^*)) = L(r_1)^*$.

Parentheses in regular expressions may be omitted, subject to the following precedence rules: $*$ has the highest precedence, then concatenation, then $+$. It is known that regular expressions define exactly the regular languages.

2.3 Grammars

We now turn to three classes of languages defined by grammars: *context-free languages* (CFLs), *linear context-free languages* (LCFLs) and *context-sensitive languages* (CSLs). These classes are denoted CF, LCF, and CS, respectively. While we describe them formally, it will suffice to note the following well-known inclusions, all of which are proper:

$$\text{REG} \subsetneq \text{LCF} \subsetneq \text{CF} \subsetneq \text{CS}. \tag{2.1}$$

For each of CF, LCF, CS, a grammar is a four-tuple $G = (V, \Sigma, P, S)$, where V is a finite set of non-terminals, Σ is a finite alphabet, $P \subseteq ((V \cup \Sigma)^* V (V \cup \Sigma)^*) \times (V \cup \Sigma)^*$ is a finite set of productions, and $S \in V$ is a distinguished start non-terminal. If $(\alpha, \beta) \in P$, we usually denote this by $\alpha \to \beta$.

Such a grammar is a *context-free grammar* (CFG) if $P \subseteq V \times (V \cup \Sigma)^*$, a *linear context-free grammar* (LCFG) if $P \subseteq V \times (\Sigma^* V \Sigma^* \cup \Sigma^*)$, and a *context-sensitive grammar* (CSG) if $(\alpha, \beta) \in P$ implies $\alpha = \eta A \zeta$ and $\beta = \eta \gamma \zeta$ for some $\eta, \zeta, \gamma \in (V \cup \Sigma)^*$, with $\gamma \neq \epsilon$ and $A \in V$.

IF $G = (V, \Sigma, P, S)$ is a grammar (CFG, LCFG or CSG), then given two words $\alpha, \beta \in (V \cup \Sigma)^*$, we denote $\alpha \Rightarrow_G \beta$ if $\alpha = \alpha_1 \alpha_2 \alpha_3$, $\beta = \alpha_1 \beta_2 \alpha_3$ for $\alpha_1, \alpha_2, \alpha_3, \beta_2 \in (V \cup \Sigma)^*$ and $\alpha_2 \to \beta_2 \in P$. Let $\Rightarrow_G^*$ denote the reflexive, transitive closure of $\Rightarrow_G$. Then the language generated by a grammar $G = (V, \Sigma, P, S)$ is given by

$$L(G) = \{x \in \Sigma^* \ : \ S \Rightarrow_G^* x\}.$$

If a language is generated by a CFG (resp., LCFG, CSG), then it is a CFL (resp., LCFL, CSL).

2.4 Complexity Theory

We now consider Turing machines. Our presentation is largely based on Hopcroft and Ullman [68, Ch. 7]. A Turing machine (TM) is a seven-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where Q is a finite set of states, Γ is a finite *tape alphabet*, $B \in \Gamma$ is the blank symbol, $\Sigma \subseteq \Gamma - B$ is the *input alphabet*, δ is the transition function given by $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, $q_0 \in Q$ is the start state, and $F \subseteq Q$ is the set of final states. This model of TM is deterministic. A nondeterministic variant is also possible.

Given a TM M, an instantaneous description (ID) of M is a word $w_1 q w_2 \in \Gamma^* Q \Gamma^*$. We interpret the ID as meaning that the TM is in state q with tape contents $w_1 w_2$ and the head currently positioned on the first character of w_2. We define a relation $\Rightarrow_M$ on the set of IDs as follows: given IDs $w_1 q_1 w_2$, $u_1 q_2 u_2$,

$$w_1 q_1 w_2 \Rightarrow_M u_1 q_2 u_2 \iff \begin{cases} \quad u_1 = w_1 \gamma,\ w_2 = \beta u_2, & \text{and } \delta(q_1, \beta) = (q_2, \gamma, R), \\ \text{or} \quad w_1 = u_1 \gamma,\ w_2 = \alpha w_2',\ u_2 = \gamma \beta w_2' & \text{and } \delta(q_1, \alpha) = (q_2, \beta, L). \end{cases}$$

Let $\Rightarrow_M^*$ be the transitive and reflexive closure of $\Rightarrow_M$. The language accepted by M, denoted $L(M)$ is

$$L(M) = \{w \in \Sigma^* \ : \ q_0 w \Rightarrow_M^* \alpha_1 q \alpha_2 \text{ such that } q \in F, \alpha_1, \alpha_2 \in \Gamma^*\}.$$

Given a language L, if there exists a TM M such that $L = L(M)$, we say that L is *recursively enumerable* (r.e.). We denote the set of r.e. languages by RE.

Say that a TM M *halts* on input x if it eventually reaches an ID which has no next move, i.e., the current ID has no successors under $\Rightarrow_M$. We may assume without loss of generality that when a word is accepted by M, M halts. However, if an input word is not accepted, we note that M may not halt.

If L is accepted by a TM M such that M halts on all inputs, we say that L is *recursive*. The set of all recursive languages is denoted by REC. The inclusions (2.1) may be extended as follows (again, the inclusions below are proper):

$$\text{CS} \subsetneq \text{REC} \subsetneq \text{RE}.$$

Nondeterminism does not affect the classes REC and RE.

We now refine the class of languages computed by a Turing machine. Given a TM M, we say that M uses space c on input w if M scans at most c tape cells during the computation on w, i.e., $\max\{|v| \ : \ q_0 w \Rightarrow_M^* vqu, w, v, u \in \Gamma^*\} \leq c$. Let n be the length of the input to a TM (i.e., the length of the word w such that $q_0 w$ is the initial ID of the TM). If a deterministic (resp., nondeterministic) TM uses at most $O(s(n))$ space on any input of length n, then we say that the language $L(M)$ is in $\text{DSPACE}(s)$ (resp., $\text{NSPACE}(s)$). It is known that $\text{CS} = \text{NSPACE}(n)$, i.e., the context-sensitive languages correspond exactly to the class of languages accepted in linear space by a nondeterministic TM. We similarly define the classes $\text{DTIME}(f)$ and $\text{NTIME}(f)$.

The following classes are also useful to us:

$$\begin{aligned} \text{P} &= \bigcup_{k \geq 1} \text{DTIME}(n^k); \\ \text{NP} &= \bigcup_{k \geq 1} \text{NTIME}(n^k). \end{aligned}$$

Given a function $g : \Sigma^* \to \Sigma^*$, we say that g is *computable* in $\text{DSPACE}(s)$ (resp., $\text{NSPACE}(s)$, $\text{DTIME}(f)$, $\text{NTIME}(f)$) if there exists a TM M operating in $\text{DSPACE}(s)$ (resp., $\text{NSPACE}(s)$, $\text{DTIME}(f)$, $\text{NTIME}(f)$) such that for all $w \in \Sigma^*$, $q_0 w \Rightarrow_M^* u_1 q u_2$ with $q \in F$ and $u_1 u_2 = g(w)$. Further, $g(w)$ is the only such tape contents which results from halting on input w.

A function $f : \mathbb{N} \to \mathbb{N}$ is said to be *space-constructible* if there exists a TM M such that $L(M) \in \text{NSPACE}(f)$ and, for all $n \geq 0$, there exists some $x \in \Sigma^n$ such that M uses exactly $f(|x|)$ space on input x.

Given two languages L', L, we say that L' is *reducible* to L if there exists a function $g : \Sigma^* \to \Sigma^*$ such that $x \in L'$ if and only if $g(x) \in L$. If g is computable in $\text{DSPACE}(\log)$, then we say that L' is log-space reducible to L.

Let $\mathcal{C}$ be a class of languages. The language L is *$\mathcal{C}$-hard* if L' is reducible to L for all $L' \in \mathcal{C}$. The language L is *$\mathcal{C}$-complete* if $L \in \mathcal{C}$ and L is $\mathcal{C}$-hard. For both P and NP, completeness can be defined with respect to log-space reductions.

2.5 Decidability

In this section, we briefly describe the concept of decidability and undecidability, and recall the Post correspondence problem (PCP) and several meta-theorems for proving undecidability.

We will often consider problems when discussing undecidability. A *problem* P is simply a predicate, in the following sense: "given an input x, does $P(x)$ hold?" For example, if P is the problem of primality, and x is an integer (encoded over our alphabet Σ), $P(x)$ holds if and only if x is a prime number. Thus, if x is suitably encoded over an alphabet Σ, P naturally defines a language over Σ^*, namely, those x such that $P(x)$ holds. Let L_P be this corresponding language (we sometimes simply identify P with the corresponding language, and do not use the notation L_P). We say that a problem P is *decidable* if $L_P \in \mathrm{REC}$. Otherwise, P is said to be *undecidable*.

The Post correspondence problem (PCP) is a basic undecidable problem which is often useful in many language-theoretic situations. An *instance* of PCP is

$$M = (u_1, u_2, \ldots, u_n; v_1, v_2, \ldots, v_n)$$

where $n \geq 1$ and $u_i, v_i \in \Sigma^*$ for $1 \leq i \leq n$. A *solution* to M is a list $i_1, i_2, \ldots, i_m$ such that $m \geq 1$, $1 \leq i_j \leq n$ for all $1 \leq j \leq m$ and

$$\prod_{j=1}^{m} u_{i_j} = \prod_{j=1}^{m} v_{i_j}.$$

The following result states that finding solutions to a PCP instance is undecidable [68, Thm. 8.8]:

Theorem 2.5.1 *Given an alphabet Σ and a PCP instance $M = (u_1, \ldots, u_n; v_1, \ldots, v_n)$, where $n \geq 1$ $u_i, v_i \in \Sigma^*$ for $1 \leq i \leq n$, it is undecidable whether there is a solution for M.*

We will also use the following undecidability result:

Theorem 2.5.2 *Let Σ be an alphabet with $|\Sigma| \geq 2$ and $G = (V, \Sigma, P, S)$ be an LCFG. It is undecidable whether $L(G) = \Sigma^*$.*

In what follows, a *predicate* on 2^{Σ^*} is simply a class of languages satisfying some property. By a predicate on a class of languages $\mathcal{C}$, we simply mean the restriction of the predicate from 2^{Σ^*}

to $\mathcal{C}$. If P is a predicate and a language $L \subseteq \Sigma^*$ satisfies P, we will denote this fact by $P(L)$. For example, if P_R is the predicate defined by the regular languages, then $P_R(L)$ implies that L is regular. A predicate P on $\mathcal{C}$ is *non-trivial* if $P \notin \{\emptyset, \mathcal{C}\}$.

Meta-theorems are powerful tools for proving undecidability. In this thesis, we will appeal to the following meta-theorem, due to Hunt and Rosenkrantz [70, Thm. 2.10], which will allow us to prove undecidability results for LCF.

Theorem 2.5.3 *Let P be a predicate on* LCF *over Σ^* such that $P(\Sigma^*)$ holds and either of the sets*

$$\{L' \ : \ L' = x \setminus L, x \in \Sigma^+, L \in \text{LCF} \textit{ and } P(L)\}$$

or

$$\{L' \ : \ L' = L/x, x \in \Sigma^+, L \in \text{LCF} \textit{ and } P(L)\}$$

is a proper subset of LCF. *Then given an LCFG G, it is undecidable whether $P(L(G))$ holds.*

The following is a corollary of Theorem 2.5.3. It is also a particular case of Greibach's Theorem (see, e.g., Hopcroft and Ullman [68, Thm. 8.14]).

Corollary 2.5.4 *Let P be a non-trivial predicate on* LCF *over Σ^* such that $P(\Sigma^*)$ holds and P is preserved under quotient. Then given an LCFG G, it is undecidable whether $P(L(G))$ holds.*

2.6 Families of Languages

We will require some definitions and notations relating to classes of languages. Let $\mathcal{C}_1, \mathcal{C}_2$ be classes of languages. Then let

$$\begin{aligned} \mathcal{C}_1 \wedge \mathcal{C}_2 &= \{L_1 \cap L_2 \ : \ L_i \in \mathcal{C}_i, i = 1, 2\}; \\ \text{co-}\mathcal{C}_1 &= \{\overline{L} \ : \ L \in \mathcal{C}_1\}. \end{aligned}$$

Our notation $\wedge$ comes from Ginsburg [51], and should not be confused with $\mathcal{C}_1 \cap \mathcal{C}_2 = \{L \ : \ L \in \mathcal{C}_1 \text{ and } L \in \mathcal{C}_2\}$.

Recall that a *cone* (or *full trio*) is a class of languages closed under morphism, inverse morphism and intersection with regular languages [148, Sect. 3].

We will also use the notion of *immune* languages. Let $\mathcal{C}$ be a class of languages. A language L is said to be $\mathcal{C}$-*immune* if L is infinite and for all infinite languages $L' \subseteq L$, $L' \notin \mathcal{C}$. Immunity was introduced for classes of languages by Flajolet and Steyaert [49]; we also refer the interested reader to Balcázar *et al.* [14] for an introduction to immunity as it relates to complexity theory.

2.7 Shuffle on Trajectories

The shuffle on trajectories operation is a method for specifying the ways in which two input words may be merged, while preserving the order of symbols in each word, to form a result. Each trajectory $t \in \{0, 1\}^*$ with $|t|_0 = n$ and $|t|_1 = m$ specifies the manner in which we can form the shuffle on trajectories of two words of length n (as the left input word) and m (as the right input word). The word resulting from the shuffle along t will have a letter from the left input word in position i if the i-th symbol of t is 0, and a letter from the right input word in position i if the i-th symbol of t is 1.

We now give the definition of shuffle on trajectories, originally due to Mateescu *et al.* [147]. Shuffle on trajectories is defined by first defining the shuffle of two words x and y over an alphabet Σ on a trajectory t, a word over $\{0, 1\}$. We denote the shuffle of x and y on trajectory t by $x \sqcup\!\sqcup_t y$.

If $x = ax'$, $y = by'$ (with $a, b \in \Sigma$) and $t = et'$ (with $e \in \{0, 1\}$), then

$$x \sqcup\!\sqcup_{et'} y = \begin{cases} a(x' \sqcup\!\sqcup_{t'} by') & \text{if } e = 0; \\ b(ax' \sqcup\!\sqcup_{t'} y') & \text{if } e = 1. \end{cases}$$

If $x = ax'$ ($a \in \Sigma$), $y = \epsilon$ and $t = et'$ ($e \in \{0, 1\}$), then

$$x \sqcup\!\sqcup_{et'} \epsilon = \begin{cases} a(x' \sqcup\!\sqcup_{t'} \epsilon) & \text{if } e = 0; \\ \emptyset & \text{otherwise.} \end{cases}$$

If $x = \epsilon$, $y = by'$ ($b \in \Sigma$) and $t = et'$ ($e \in \{0, 1\}$), then

$$\epsilon \sqcup\!\sqcup_{et'} y = \begin{cases} b(\epsilon \sqcup\!\sqcup_{t'} y') & \text{if } e = 1; \\ \emptyset & \text{otherwise.} \end{cases}$$

We let $x \sqcup\!\sqcup_{\epsilon} y = \emptyset$ if $\{x, y\} \neq \{\epsilon\}$. Finally, if $x = y = \epsilon$, then $\epsilon \sqcup\!\sqcup_t \epsilon = \epsilon$ if $t = \epsilon$ and $\emptyset$ otherwise.

It is not difficult to see that if $t = \prod_{i=1}^{n} 0^{j_i} 1^{k_i}$ for some $n \geq 0$ and $j_i, k_i \geq 0$ for all $1 \leq i \leq n$, then we have that

$$x \sqcup\!\sqcup_t y = \{\prod_{i=1}^{n} x_i y_i \ : \ x = \prod_{i=1}^{n} x_i,\, y = \prod_{i=1}^{n} y_i, \text{ with } |x_i| = j_i, |y_i| = k_i \text{ for all } 1 \leq i \leq n\}$$

if $|x| = |t|_0$ and $|y| = |t|_1$ and $x \sqcup\!\sqcup_t y = \emptyset$ if $|x| \neq |t|_0$ or $|y| \neq |t|_1$.

We extend shuffle on trajectories to *sets* $T \subseteq \{0, 1\}^*$ *of trajectories* as follows:

$$x \sqcup\!\sqcup_T y = \bigcup_{t \in T} x \sqcup\!\sqcup_t y.$$

Further, for $L_1, L_2 \subseteq \Sigma^*$, we define

$$L_1 \sqcup\!\sqcup_T L_2 = \bigcup_{\substack{x \in L_1 \\ y \in L_2}} x \sqcup\!\sqcup_T y.$$

2.7.1 Examples

We now consider some examples of shuffle on trajectories. Let $x = abc$ and $y = de$. If $t = 00011$, then $x \sqcup\!\sqcup_t y = abcde$. If $t = 00111$, then $x \sqcup\!\sqcup_t y = \emptyset$. Thus, we can see that if $T = 0^*1^*$, we have that

$$L_1 \sqcup\!\sqcup_T L_2 = L_1 L_2,$$

i.e., $T = 0^*1^*$ gives the concatenation operation.

If $x = abc$, $y = de$, and $t = 01001$, then $x \sqcup\!\sqcup_t y = adbce$. If $t = 01010$, then $x \sqcup\!\sqcup_t y = adbec$. Thus, we have that if $T = (0 + 1)^*$, then

$$L_1 \sqcup\!\sqcup_T L_2 = L_1 \sqcup\!\sqcup L_2,$$

i.e., $T = \{0, 1\}^*$ gives the shuffle operation. This is the least restrictive set of trajectories.

If $T = 0^*1^*0^*$, then $\sqcup\!\sqcup_T$ is the insertion operation $\leftarrow$ (see, e.g, Kari [106]) which is defined by $x \leftarrow y = \{x_1 y x_2 \ : \ x_1, x_2 \in \Sigma^*, x_1 x_2 = x\}$ for all $x, y \in \Sigma^*$. Some other examples of operations defined by shuffle on trajectories are given in Figure 2.2 in the following section.

2.7.2 Algebraic Properties

We will require some algebraic properties of shuffle on trajectories throughout this thesis. These properties have been studied by Mateescu *et al.* [147].

Let $T \subseteq \{0, 1\}^*$. We say that T is *complete* if, for all $x, y \in \Sigma^*$, $x \sqcup\!\sqcup_T y \neq \emptyset$, i.e., there exists some $z \in \Sigma^*$ such that $z \in x \sqcup\!\sqcup_T y$. The set T is said to be *deterministic* if, for all $x, y \in \Sigma^*$, $|x \sqcup\!\sqcup_T y| \leq 1$. Say that T is *associative* (resp., *commutative*) if the corresponding operation $\sqcup\!\sqcup_T$ is associative (resp., commutative), i.e., $x \sqcup\!\sqcup_T (y \sqcup\!\sqcup_T z) = (x \sqcup\!\sqcup_T y) \sqcup\!\sqcup_T z$ for all $x, y, z \in \Sigma^*$ (resp., $x \sqcup\!\sqcup_T y = y \sqcup\!\sqcup_T x$ for all $x, y \in \Sigma^*$). For characterizations and decidability of these properties, we refer the reader to Mateescu *et al.* [147, Sect. 4]. We summarize several examples of shuffle on trajectories and their algebraic properties in Figure 2.2.

Name	T	Complete?	Determ.?	Assoc.?	Commutative?
Concatenation	0^*1^*	$\surd$	$\surd$	$\surd$	$\times$
Insertion	$0^*1^*0^*$	$\surd$	$\times$	$\times$	$\times$
Shuffle	$(0+1)^*$	$\surd$	$\times$	$\surd$	$\surd$
Perfect Shuffle	$(01)^*$	$\times$	$\surd$	$\times$	$\times$
Balanced Insertion	$\{0^i 1^{2j} 0^i \ : i, j \geq 0\}$	$\times$	$\surd$	$\surd$	$\times$
Bi-catenation	$0^*1^* + 1^*0^*$	$\surd$	$\times$	$\times$	$\surd$

Figure 2.2: Some examples of shuffle on trajectories and their algebraic properties.