Chapter 5

Automata, Circuits, and Logic

Although RNNs aligned fairly well with various automata that are taught in *Theory of Computing* or equivalent undergraduate courses, the same is not true of transformers. So we need to introduce some new concepts. In this chapter, we'll learn about a subclass of regular languages called the star-free regular languages Section 5.1, which can be characterized by subclasses of finite automata and regular expressions, as well as by two logics, first-order logic and linear temporal logic. Then, we'll learn about a new model of computation, Boolean circuits (Section 5.2). Finally, we'll learn about some extensions to first-order logic (Section 5.3).

5.1 Star-free regular languages

Star-free languages are called that because they are described by star-free regular expressions. Although we won't use star-free regular expressions later, we might as well explain where the name comes from.

The syntax of star-free regular expressions over a finite alphabet Σ is defined in BNF as:

$$\begin{aligned} \alpha ::= {} & \emptyset \\ & \mid \epsilon \\ & \mid \sigma \qquad\qquad \sigma \in \Sigma \\ & \mid \alpha_1 \cup \alpha_2 \\ & \mid \alpha_1 \alpha_2 \\ & \mid \alpha^{\mathsf{C}} \end{aligned} \tag{5.1}$$

Example 5.1. Let $\Sigma = \{\mathsf{a}, \mathsf{b}\}$.

- Σ^* is star-free because $\Sigma^* = \emptyset^{\mathsf{C}}$.
- $(\mathsf{ab})^*$ is star-free because $(\mathsf{ab})^* = (\mathsf{b}\Sigma^* \cup \Sigma^*\mathsf{a} \cup \Sigma^*\mathsf{aa}\Sigma^* \cup \Sigma^*\mathsf{bb}\Sigma^*)^{\mathsf{C}}$.
- $(\mathsf{aa})^*$ is regular but not star-free.

For more on star-free languages, see the survey by Pin (2020).

5.1.1 Counter-free automata

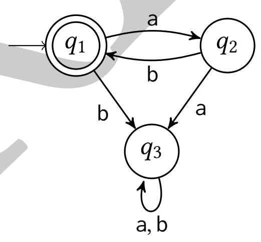
Definition 5.2. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Define the relation $q \xrightarrow{w} r$, where $q, r \in Q$ and $w \in \Sigma^*$, to mean "If M is in state q and reads string w, then it ends up in state q." That is:

- $q \xrightarrow{\epsilon} q$
- $q \xrightarrow{av} s$ iff, for some r, $\delta(q, a) = r$ and $r \xrightarrow{v} s$.

We say that M is *counter-free* iff there is an N such that for all $w \in \Sigma^*$ and $n \geq N$, the relations $\xrightarrow{w^n}$ and $\xrightarrow{w^{n+1}}$ are the same.

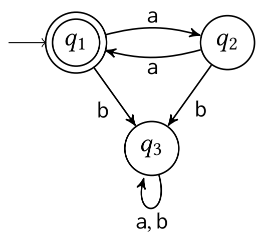
Example 5.3. Intuitively, a counter-free DFA is one that can test whether something happens, but not how many times it happens. For every cycle $q \xrightarrow{w} q$, w cannot be written as x^k where $x \in \Sigma^*$ and $k > 1$.

(a) The following DFA, which recognizes $(\mathsf{ab})^*$, is counter-free:



This DFA is counter-free, because the only cycles are on ab (from q_1 to itself), a and b (from q_3 to itself), and none of these strings is of the form x^k for $k > 1$.

(b) The following DFA, which recognizes $(\mathsf{aa})^*$, is not counter-free:



This is not counter-free, because it has a cycle on aa, which is a^2.

Theorem 5.4 (Schützenberger, 1965; McNaughton and Papert, 1971)**.** *A language L is star-free regular if and only if its minimal DFA is counter-free.*

So in Example 5.3(b), the fact that the DFA shown is minimal and not counter-free shows that (aa∗) is not star-free.

5.1.2 First-order logic

A formal language can also be defined as a set of finite strings that satisfy a closed formula of a logic. This chapter is a very expanded version of Section 5.3 of the survey by Strobl et al. (2024); for more information, see the handbook chapter by Thomas (1997).

Although it will not become important until later, we number the positions of a string starting from 0, so $w = w_0 w_1 \cdots w_{n-1}$.

Example 5.5. Let $\Sigma = \{\mathrm{a}, \mathrm{b}\}$. The formula

$$\phi = \forall x.\forall y.Q_{\mathrm{a}}(x) \land Q_{\mathrm{b}}(y) \rightarrow x < y \tag{5.2}$$

defines the regular language $\mathrm{a}^*\mathrm{b}^*$. The variables (x, y) are interpreted as positions of a string w, and $Q_{\mathrm{a}}(i)$ is true iff $w_i = \mathrm{a}$ and $Q_{\mathrm{b}}(i)$ is true iff $w_i = \mathrm{b}$. So the formula says that every a must precede every b, which is true iff the string matches $\mathrm{a}^*\mathrm{b}^*$.

We first define the syntax of first-order logic with order (FO).

Definition 5.6. The *formulas* of FO are given by the BNF grammar:

$$\begin{aligned} \phi ::= {} & Q_\sigma(x) && \sigma \in \Sigma \\ & \mid x = y \mid x < y \\ & \mid \phi_1 \land \phi_2 \mid \phi_1 \lor \phi_2 \mid \neg\phi_1 \\ & \mid \forall x.\phi_1 \mid \exists x.\phi_1 \end{aligned} \tag{5.3}$$

where $x, y, \ldots$ are variables. The *free variables* of a formula are defined as follows:

$$\begin{aligned}
\mathrm{FV}(Q_\sigma(x)) &= \{x\} && \sigma \in \Sigma \\
\mathrm{FV}(x = y) &= \{x, y\} \\
\mathrm{FV}(x < y) &= \{x, y\} \\
\mathrm{FV}(\phi_1 \wedge \phi_2) &= \mathrm{FV}(\phi_1) \cup \mathrm{FV}(\phi_2) \\
\mathrm{FV}(\phi_1 \vee \phi_2) &= \mathrm{FV}(\phi_1) \cup \mathrm{FV}(\phi_2) \\
\mathrm{FV}(\neg\phi_1) &= \mathrm{FV}(\phi_1) \\
\mathrm{FV}(\forall x.\phi_1) &= \mathrm{FV}(\phi_1) \setminus \{x\} \\
\mathrm{FV}(\exists x.\phi_1) &= \mathrm{FV}(\phi_1) \setminus \{x\}.
\end{aligned} \tag{5.4}$$

A formula is *closed* if it has no free variables. For example, $\mathrm{FV}(\exists y.x < y) = \{x\}$, while the formula in Eq. (5.2) is closed.

We use a number of shorthand notations:

$$\phi_1 \rightarrow \phi_2 = \neg\phi_1 \vee \phi_2 \qquad \text{implication} \tag{5.5}$$

$$\phi_1 \leftrightarrow \phi_2 = (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1) \qquad \text{if and only if} \tag{5.6}$$

$$\phi_1 \oplus \phi_2 = \neg(\phi_1 \leftrightarrow \phi_2) \qquad \text{exclusive or} \tag{5.7}$$

The semantics of FO defines whether formulas are true in various *logical structures*. Here, we are only interested in logical structures that correspond to strings. So we skip the definition of logical structure and go straight to the definition of whether a formula is true for a string.

Definition 5.7. Let $w = w_0 \cdots w_{n-1}$ be a string over Σ, and let I be an *interpretation*, or a mapping from variables to $\{0, \ldots, n-1\}$. We define $w, I \models \phi$ ("w and I satisfy ϕ") as follows:

$$\begin{aligned}
&w, I \models Q_\sigma(x) && \text{if } w_{I(x)} = \sigma \\
&w, I \models x = y && \text{if } I(x) = I(y) \\
&w, I \models x < y && \text{if } I(x) < I(y) \\
&w, I \models \phi_1 \wedge \phi_2 && \text{if } w, I \models \phi_1 \text{ and } w, I \models \phi_2[I] \\
&w, I \models \phi_1 \vee \phi_2 && \text{if } w, I \models \phi_1 \text{ or } w, I \models \phi_2 \\
&w, I \models \neg\phi_1 && \text{if } w, I \not\models \phi_1 \\
&w, I \models \forall x.\phi_1 && \text{if } w, I[x \mapsto i] \models \phi_1 \text{ for all } i \in \{0, \ldots, n-1\} \\
&w, I \models \exists x.\phi_1 && \text{if } w, I[x \mapsto i] \models \phi_1 \text{ for some } i \in \{0, \ldots, n-1\}
\end{aligned} \tag{5.8}$$

Above, the notation $I[x \mapsto i]$ stands for the mapping that sends x to i, and sends any other variable y to $I(y)$.

If $w, I \models \phi$, and ϕ is a closed formula, we can simply write $w \models \phi$. The language defined by a closed formula ϕ is $L(\phi) = \{w \in \Sigma^* \mid w \models \phi\}$.

Example 5.8. Let ϕ be as in Eq. (5.2). Here are some examples of strings and interpretations that do or don't satisfy ϕ:

$$\begin{aligned}
\mathsf{abb}, \{x \mapsto 0\} &\models Q_\mathsf{a}(x) \\
\mathsf{abb}, \{y \mapsto 0\} &\not\models Q_\mathsf{b}(y) \\
\mathsf{abb}, \{y \mapsto 1\} &\models Q_\mathsf{b}(y) \\
\mathsf{abb}, \{y \mapsto 2\} &\models Q_\mathsf{b}(y) \\
\mathsf{abb}, \{x \mapsto 0, y \mapsto 0\} &\not\models x < y \\
\mathsf{abb}, \{x \mapsto 0, y \mapsto 1\} &\models x < y \\
\mathsf{abb}, \{x \mapsto 0, y \mapsto 2\} &\models x < y \\
\mathsf{abb}, \{x \mapsto 0, y \mapsto 0\} &\models Q_\mathsf{a}(x) \land Q_\mathsf{b}(y) \to x < y \\
\mathsf{abb}, \{x \mapsto 0, y \mapsto 1\} &\models Q_\mathsf{a}(x) \land Q_\mathsf{b}(y) \to x < y \\
\mathsf{abb}, \{x \mapsto 0, y \mapsto 2\} &\models Q_\mathsf{a}(x) \land Q_\mathsf{b}(y) \to x < y \\
\mathsf{abb}, \{x \mapsto 0\} &\models \forall y. Q_\mathsf{a}(x) \land Q_\mathsf{b}(y) \to x < y
\end{aligned}$$

Example 5.9. As a slightly more complicated example of what can be defined in FO, let

$$\text{FIRST}(x) = \neg\exists y. y < x \tag{5.9}$$
$$\text{SUCC}(x, y) = x < y \land \neg\exists z. x < z < y \tag{5.10}$$
$$\text{LAST}(x) = \neg\exists y. y > x \tag{5.11}$$

Then

$$\begin{aligned}
\phi = {} & (\forall x. \text{FIRST}(x) \to Q_\mathsf{a}(x)) \\
& \land (\forall x. \forall y. \text{SUCC}(x, y) \to ((Q_\mathsf{a}(x) \land Q_\mathsf{b}(y)) \lor (Q_\mathsf{b}(x) \land Q_\mathsf{a}(y)))) \\
& \land (\forall y. \text{LAST}(x) \to Q_\mathsf{b}(x))
\end{aligned} \tag{5.12}$$

defines the language $(\mathsf{ab})^*$.

Theorem 5.10 (McNaughton and Papert, 1971)**.** FO *defines exactly the class of star-free regular languages.*

5.1.3 Linear temporal logic

In *linear temporal logic* (Kamp, 1968), every formula implicitly depends on a single time (or position). Here, we use the variant called *past temporal logic* or PTL, which is as expressive as the full version (Gabbay et al., 1980).

Example 5.11. Let $\Sigma = \{\mathsf{a}, \mathsf{b}, \#\}$.

- The formula

$$\phi = Q_{\#} \tag{5.13}$$

defines the language $\Sigma^*\#$, which contains all and only strings with a # in the *last* position.

- The formula

$$\phi = Q_{\#} \wedge (Q_{\mathsf{b}} \textbf{ since } Q_{\#}) \tag{5.14}$$

defines the language $\Sigma^*\#\mathsf{b}^*\#$. ("The last symbol is #, and ever since the previous #, it's been all b's.")

- The formula

$$\phi = Q_{\#} \wedge (Q_{\mathsf{b}} \textbf{ since } (Q_{\#} \wedge (Q_{\mathsf{a}} \textbf{ since } Q_{\#}))) \tag{5.15}$$

defines the language $\Sigma^*\#\mathsf{a}^*\#\mathsf{b}^*\#$.

The syntax of PTL is defined as follows:

$$\begin{aligned} \phi ::= {} & Q_\sigma && \sigma \in \Sigma \\ & \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi_1 \\ & \mid \phi_1 \textbf{ since } \phi_2 \end{aligned} \tag{5.16}$$

For any input string $w = w_0 \cdots w_{n-1}$ and position $i \in \{0, \ldots, n-1\}$, we define $w, i \models \phi$ as follows:

$$\begin{aligned} & w, i \models Q_\sigma && \text{if } w_i = \sigma \\ & w, i \models \phi_1 \wedge \phi_2 && \text{if } w, i \models \phi_1 \text{ and } w, i \models \phi_2 \\ & w, i \models \phi_1 \vee \phi_2 && \text{if either } w, i \models \phi_1 \text{ or } w, i \models \phi_2 \\ & w, i \models \neg\phi_1 && \text{if } w, i \not\models \phi_1 \\ & w, i \models \phi_1 \textbf{ since } \phi_2 && \text{if there is a } j < i \text{ such that } w, j \models \phi_2 \text{, and} \\ & && \quad \text{for all } k \text{ such that } j < k < i \text{, we have } w, k \models \phi_1 \end{aligned} \tag{5.17}$$

To use a formula ϕ of PTL to define a language over Σ, for a $w \in \Sigma^*$ of length n we supply w as input and designate the last position as the output position, so that $w \in \mathcal{L}(\phi)$ if and only if $w, n \models \phi$.

Theorem 5.12 (Kamp 1968; Gabbay et al. 1980)**.** PTL *defines the exactly the class of star-free regular languages.*

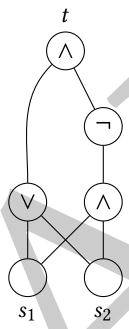
5.2 Circuit complexity

This section is an expanded version of Section 5.2 of the survey by Strobl et al. (2024). For more details, please see the textbook by Arora and Barak (2009).

For the rest of this chapter, we deal a lot with bits. We write $\log n$ for $\lceil \log_2 n \rceil$, which is the number of bits needed to represent a number in $\{0, \ldots, n-1\}$.

Circuits operate on binary values, so for the rest of this section, we assume $\Sigma = \{0, 1\}$. (If we want to use circuits to model sets of strings over an alphabet Σ, we can choose a fixed-length encoding of the symbols of Σ as strings of $b = \log |\Sigma|$ bits and encode the value of the i-th input symbol into positions ib to $ib + (b-1)$.)

Example 5.13. Here's a circuit with input length 2. It computes the XOR function. We draw the inputs at the bottom and the output at the top.



Definition 5.14. A *(Boolean) circuit* C with input length n is a directed acyclic graph with n *input* nodes $s_1, \ldots, s_n$, with fan-in (in-degree) zero, and zero or more *gate* nodes, each labeled with one of the following functions:

- NOT ($\neg$), with fan-in one.
- AND ($\wedge$), with arbitrary fan-in. (An AND gate with fan-in zero always has value 1.)
- OR ($\vee$), with arbitrary fan-in. (An OR gate with fan-in zero always has value 0.)

One node t, which can be either an input or gate node, is designated the *output* of the circuit.

Given an input string $w \in \{0, 1\}^n$, each input node s_i is assigned the value w_i, and each gate node labeled f computes its value by applying f to the values of its in-neighbors. Thus we can think of the circuit as computing a Boolean function $C \colon \{0, 1\}^n \to \{0, 1\}$, mapping each input string to the value of t. The *depth* of C, denoted $\operatorname{depth}(C)$, is the length of the longest directed path from any s_i to t. The *size* of C, denoted $|C|$, is the number of nodes in C.

Definition 5.15. A *circuit family* is a sequence $C = (C_n)_{n\in\mathbb{N}}$ such that for each n, C_n is a circuit with input length n. We treat C as a function on $\{0,1\}^*$ as follows: for every $w \in \{0,1\}^*$ with length n, $C(w) = C_n(w)$. Then the language defined by C is $L(C) = \{w \in \{0,1\}^* \mid C(w) = 1\}$. The *depth* and *size* of C are the functions $n \mapsto \text{depth}(C_n)$ and $n \mapsto |C_n|$.

Since the depth and size of a circuit family are functions, we are interested in how they depend asymptotically on n. In particular, since transformers have constant depth, circuit classes with constant depth are of particular interest.

Definition 5.16. We define the following classes of languages:

- AC^0 contains those languages that can be recognized by families of circuits with unbounded fan-in, $O(1)$ depth, and $O(\text{poly}(n))$ size.
- TC^0 is like AC^0, but also allows MAJORITY gates, which have unbounded fan-in and output 1 iff at least half of their inputs are 1.
- NC^1 is like AC^0, but with fan-in at most 2 and depth in $O(\log n)$.

A circuit family contains a different circuit for each length n, with no constraint on the relationship between the circuits. This has possibly surprising consequences.

Example 5.17. Let L be any *unary* language, that is, $L \subseteq \{1\}^*$. For each $n \in \mathbb{N}$, if $1^n \in L$, let C_n be a circuit that always has value 1 (an AND gate with fan-in zero), and if $1^n \notin L$, let C_n be a circuit that has value 0 (an OR gate with fan-in zero). Thus, L is recognized by a circuit family with $O(n)$ size and $O(1)$ depth, and is therefore in AC^0, even if it is undecidable.

To prevent such consequences, we impose a $\mathsf{DLOGTIME}$-*uniform* restriction, which says that, given n, the circuit C_n can be constructed in logarithmic time, in the following sense.

Definition 5.18 (Barrington et al., 1990)**.** Let $C = (C_n)_{n\in\mathbb{N}}$ be a circuit family, and assume that the nodes of C_n are numbered from 0 to $|C_n| - 1$. We say that C is $\mathsf{DLOGTIME}$-*uniform* if there is a (deterministic) Turing machine that runs in logarithmic time and accepts those tuples $\langle f, i, j, 1^n\rangle$ such that in C_n, node i has label f and there is an edge from node i to node j.

5.3 More Logic

5.3.1 Arithmetic predicates

We can increase the expressivity of FO by adding more predicates besides $<$. The logic $\mathsf{FO}[\text{BIT}]$ extends FO with a predicate $\text{BIT}(x, y)$, which tests whether the y-th bit (that is, the one with place value 2^y) of x is set. That is, Eq. (5.8) is extended with:

$$w, I \models \mathrm{BIT}(x, y) \qquad \text{if } \left\lfloor I(x)/2^{I(y)} \right\rfloor \text{ is odd.} \tag{5.18}$$

Theorem 5.19. *The following formulas are definable in* FO[BIT]*:*

(a) $\mathrm{ADD}(x, y, z)$ *iff* $z = x + y$ *(more precisely,* $w, I \models \mathrm{ADD}(x, y, z)$ *iff* $I(z) = I(x) + I(y)$*)*

(b) $\mathrm{MUL}(x, y, z)$ *iff* $z = xy$

(c) $\mathrm{DIV}(x, y, z)$ *iff* $z = \lfloor x/y \rfloor$

(d) $\mathrm{POW}(x, y, z)$ *iff* $z = x^y$

For readability, we usually write $x + y = z$ in place of $\mathrm{ADD}(x, y, z)$, and similarly for other arithmetic operations. Note that variables are interpreted in $\{0, \ldots, n-1\}$, so if $n = 10$ then $5 + 5 = z$ is false for all z.

Example 5.20. The following formula of FO[BIT] tests whether a number is odd:

$$\mathrm{ODD}(x) = \neg(\exists y.\mathrm{ADD}(y, y, x)) = \neg(\exists y. y + y = x). \tag{5.19}$$

Theorem 5.21 (Barrington et al., 1990)**.** FO[BIT] *defines exactly the languages in* DLOGTIME*-uniform* AC^0.

5.3.2 Counting quantifiers

FOC is first order logic with *counting quantifiers* (Immerman, 1999, p. 185–187).

Example 5.22. The majority language,

$$\mathsf{MAJORITY} = \{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid w \text{ has more } \mathtt{1}\text{'s than } \mathtt{0}\text{'s}\}. \tag{5.20}$$

can be defined by the FOC formula

$$\exists x.\exists y.\ \underbrace{(\exists^{=x} z. Q_{\mathtt{0}}(z))}_{\text{there are } x \text{ many } \mathtt{0}\text{'s}} \wedge \underbrace{(\exists^{=y} z. Q_{\mathtt{1}}(z))}_{\text{there are } y \text{ many } \mathtt{1}\text{'s}} \wedge y > x. \tag{5.21}$$

The syntax of FOC is that of FO (Eq. (5.3)), as well as:

$$\phi ::= \exists^{=x} y.\phi_1 \tag{5.22}$$

We extend the definition of free variables (Eq. (5.4)) with:

$$\mathrm{FV}(\exists^{=x} y.\phi_1) = \mathrm{FV}(\phi_1) \setminus \{y\} \cup \{x\} \tag{5.23}$$

And we extend the definition of $\models$ (Eq. (5.8)) with:

$$w, I \models \exists^{=x} y.\phi_1 \qquad \text{if } |\{j \mid w, I[y \mapsto j] \models \phi_1\}| = I(x). \tag{5.24}$$

Alternatively, counting quantifiers can be defined in terms of *majority* quantifiers and vice versa. The formula $\mathsf{M}x.\phi(x)$ is true iff $\phi(x)$ is true for a (strict) majority of positions x. We write FOM for FO extended with majority quantifiers.

Example 5.23. The language MAJORITY can of course be defined in FOM, by the formula

$$\mathsf{M}x.Q_1(x). \tag{5.25}$$

Theorem 5.24. FOC *and* FOM *define exactly the same languages.*

From now on, we write FOC and FOM interchangeably.

Just as we extended FO with the BIT predicate, we can extend FOM with BIT to get FOM[BIT].

Example 5.25. PARITY (Example 4.2) can be defined by the FOM[BIT] formula

$$\exists x.\ \underbrace{(\exists^{=x} y.Q_1(y))}_{\text{there are } x \text{ many 1's}} \wedge \underbrace{\neg(\exists z.z + z = x)}_{x \text{ is odd}}. \tag{5.26}$$

Theorem 5.26 (Addition of $O(n)$ numbers with $O(\log n)$ bits)**.** *If $\phi(i,x)$ is a formula of* FOM[BIT] *such that for each $i \in \{0,\ldots,n-1\}$, $\phi(i,x)$ is true for exactly one x, then there is a formula* $\mathrm{SUM}_\phi(y)$ *which is true iff*

$$y = \sum_{\substack{i \in \{0,\ldots,n-1\} \\ x \text{ s.t. } \phi(i,x) \text{ true}}} x.$$

Proof. For each $i \in \{0,\ldots,n-1\}$, let x_i be such that $\phi(i,x_i)$ is true, and let x_{ij} be the j-th bit of x_i. Then if we picture the sum as

$$\begin{array}{rcc}
x_{1,\log n} & \cdots & x_{1,0} \\
\vdots & \ddots & \vdots \\
+\, x_{n,\log n} & \cdots & x_{n,0} \\
\hline
y_{\log n} & \cdots & y_0
\end{array}$$

then we can use a counting quantifier to sum the j-th column of bits (shifted left by j bits):

$$\mathrm{COLSUM}_\phi(j,s') = \exists s.(\exists^{=s} i.\exists x.\phi(i,x) \wedge \mathrm{BIT}(x,j)) \wedge s' = 2^j \times s. \tag{5.27}$$

Now $\mathrm{COLSUM}(j,s')$ defines $\log n$ numbers (one for each j) with $\log n$ bits each. We want their sum, which is definable in FO[BIT] (Immerman, 1999, proof of Theorem 1.17.1). $\square$

Theorem 5.27 (Barrington et al., 1990)**.** $\mathsf{FOM}[\mathsf{BIT}]$ *defines exactly the languages in* $\mathsf{DLOGTIME}$*-uniform* TC^0.

Bibliography

Arora, Sanjeev and Boaz Barak (2009). *Computational Complexity: A Modern Approach.* Cambridge University Press.

Barrington, David A. Mix, Neil Immerman, and Howard Straubing (1990). On uniformity within NC^1. In: *Journal of Computer and System Sciences* 41.3, pp. 274–306.

Gabbay, Dov, Amir Pnueli, Saharon Shelah, and Jonathan Stavi (1980). On the temporal analysis of fairness. In: *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pp. 163–173.

Immerman, Neil (1999). *Descriptive Complexity*. Springer.

Kamp, Johan Anthony Willem (1968). *Tense Logic and the Theory of Linear Order*. PhD thesis. University of California, Los Angeles.

McNaughton, Robert and Seymour A. Papert (1971). *Counter-Free Automata.* MIT Press.

Pin, Jean-Éric (2020). How to prove that a language is regular or star-free? In: *Language and Automata Theory and Applications (LATA).* Lecture Notes in Computer Science 12038, pp. 68–88.

Schützenberger, M. P. (1965). On finite monoids having only trivial subgroups. In: *Information and Control* 8.2, pp. 190–194.

Strobl, Lena, William Merrill, Gail Weiss, David Chiang, and Dana Angluin (2024). What formal languages can transformers express? A survey. In: *Transactions of the Association for Computational Linguistics.* To appear.

Thomas, Wolfgang (1997). Languages, automata, and logic. In: *Handbook of Formal Languages: Volume 3, Beyond Words.* Ed. by Grzegorz Rozenberg and Arto Salomaa. Springer, pp. 389–455.