CS5236 – Advanced Automata Theory

Frank Stephan

Semester I, Academic Year 2022-2023

Advanced Automata Theory is a lecture which will first review the basics of formal languages and automata theory and then give insight into specific topics from wider area of automata theory. In computer science, automata are an important tool for many theoretical results and various types of automata have been used to characterise complexity classes. The lecture will give a deeper understanding of automata theory, describe the Chomsky hierarchy and introduce to various advanced topics like automatic structures, automata on infinite words, automata on trees and the learnability of classes of regular languages from queries and from positive data.

Frank Stephan: Rooms S17#07-04 and COM2#03-11
Departments of Mathematics and Computer Science
National University of Singapore
10 Lower Kent Ridge Road, Singapore 119076
Republic of Singapore
Telephone 65162759 and 65164246
Email fstephan@comp.nus.edu.sg
Homepage http://www.comp.nus.edu.sg/˜fstephan/index.html

These lecture notes contain all material relevant for the examinations and the course. For further reading, the author refers to the textbooks *Automata Theory and its Applications* by Bakhadyr Khoussainov and Anil Nerode [54] and *Introduction to Automata Theory, Languages, and Computation* by Hopcroft, Motwani and Ullman [43]. Epstein, Cannon, Holt, Levy, Paterson and Thurston [28] give background on the group-theoretic parts of this lecture.

The author would like to thank Volker Diekert, Henning Fernau, Vladimir Gusev, Sanjay Jain, Bakhadyr Khoussainov, Dietrich Kuske, Alexander Lauser, Elaine Fang Li, Wolfgang Merkle, Pavel Semukhin and Mengjin Zhuang for correspondence and discussions.

Contents

1 Chomsky Hierarchy and Grammars

In theoretical computer science, one considers several main ways to describe a language L; here a language is usually a set of strings w over an alphabet Σ. The alphabet Σ is usually finite. For example, $\{\varepsilon, 01, 10, 0011, 0101, 0110, 1001, 1010, 1100, 000111, \ldots\}$ is the language of all strings over $\{0, 1\}$ which contain as many 0 as 1. Furthermore, let vw or $v \cdot w$ denote the concatenation of the strings v and w by putting the symbols of the second string behind those of the first string: $001 \cdot 01 = 00101$. Sets of strings are quite important, here some ways to define sets.

Definition 1.1. **(a)** *A finite list in set brackets denotes the set of the corresponding elements, for example* $\{001, 0011, 00111\}$ *is the set of all strings which have two* 0*s followed by one to three* 1*s.*

(b) *For any set L, let* L^* *be the set of all strings obtained by concatenating finitely many strings from L:* $L^* = \{u_1 \cdot u_2 \cdot \ldots \cdot u_n : n \in \mathbb{N} \wedge u_1, u_2, \ldots, u_n \in L\}$.

(c) *For any two sets L and H, let $L \cup H$ denote the union of L and H, that is, the set of all strings which are in L or in H.*

(d) *For any two sets L and H, let $L \cap H$ denote the intersection of L and H, that is, the set of all strings which are in L and in H.*

(e) *For any two sets L and H, let $L \cdot H$ denote the set* $\{v \cdot w : v \in L \wedge w \in H\}$, *that is, the set of concatenations of members of L and H.*

(f) *For any two sets L and H, let $L - H$ denote the set difference of L and H, that is,* $L - H = \{u : u \in L \wedge u \notin H\}$.

Remarks 1.2. For finite sets, the following additional conventions are important: The symbol $\emptyset$ is a special symbol which denotes the empty set – it could also be written as $\{\,\}$. The symbol ε denotes the empty string and $\{\varepsilon\}$ is the set containing the empty string.

In general, sets of strings considered in this lecture are usually sets of strings over a fixed alphabet Σ. Σ^* is then the set of all strings over the alphabet Σ.

Besides this, one can also consider L^* for sets L which are not an alphabet but already a set of strings themselves: For example, $\{0, 01, 011, 0111\}^*$ is the set of all strings which are either empty or start with 0 and have never more than three consecutive 1s. The empty set $\emptyset$ and the set $\{\varepsilon\}$ are the only sets where the corresponding starred set is finite: $\emptyset^* = \{\varepsilon\}^* = \{\varepsilon\}$. The operation $L \mapsto L^*$ is called the "Kleene star operation" named after Stephen Cole Kleene who introduced this notion.

An example for a union is $\{0, 11\} \cup \{01, 11\} = \{0, 01, 11\}$ and for an intersection is $\{0, 11\} \cap \{01, 11\} = \{11\}$. Note that $L \cap H = L - (L - H)$ for all sets L and H.

Formal languages are languages L for which there is a mechanism to check membership in L or to generate all members of L. The various ways to describe a language

L are given by the following types of mechanisms:

- By a mechanism which checks whether a given word w belongs to L. Such a mechanism is called an automaton or a machine.
- By a mechanism which generates all the words w belonging to L. This mechanism is step-wise and consists of rules which can be applied to derive the word in question. Such a mechanism is called a grammar.
- By a function which translates words to words such that L is the image of another (simpler) language H under this function. There are various types of functions f to be considered and some of the mechanisms to compute f are called transducers.
- An expression which describes in a short-hand the language considered like, for example, $\{01, 10, 11\}^*$. Important are here in particular the regular expressions.

Regular languages are those languages which can be defined using regular expressions. Later, various characterisations will be given for these languages. Regular expressions are a quite convenient method to describe sets.

Definition 1.3. *A regular expression denotes either a finite set (by listing its elements), the empty set by using the symbol $\emptyset$ or is formed from other regular expressions by the operations given in Definition 1.1 (which are Kleene star, concatenation, union, intersection and set difference).*

Convention. For regular expressions, one usually fixes a finite alphabet Σ first. Then all the finite sets listed are sets of finite strings over Σ. Furthermore, one does not use complement or intersection, as these operations can be defined using the other operations. Furthermore, for a single word w, one writes a^* in place of $\{a\}^*$ and abc^* in place of $\{ab\} \cdot \{c\}^*$. For a single variable w, w^* denotes $(w)^*$, even if w has several symbols. L^+ denotes the set of all non-empty concatenations over members of L; so L^+ contains ε iff L contains ε and L^+ contains a non-empty string w iff $w \in L^*$. Note that $L^+ = L \cdot L^*$. Sometimes, in regular expressions, $L + H$ is written in place of $L \cup H$. This stems from the time where typesetting was mainly done only using the symbols on the keyboard and then the addition-symbol was a convenient replacement for the union.

Example 1.4. The regular language $\{00, 11\}^*$ consists of all strings of even length where each symbol in an even position (position 0, 2, ...) is repeated in the next odd position. So the language contains 0011 and 110011001111 but not 0110.

The regular language $\{0, 1\}^* \cdot 001 \cdot \{0, 1, 2\}^*$ is the set of all strings where after some 0s and 1s the substring 001 occurs, followed by an arbitrary number of 0s and

1s and 2s.

The regular set $\{00, 01, 10, 11\}^* \cap \{000, 001, 010, 010, 100, 101, 110, 111\}^*$ consists of all binary strings whose length is a multiple of 6.

The regular set $\{0\} \cup \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \cdot \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^*$ consists of all decimal representations of natural numbers without leading 0s.

The set of binary numbers (without leading zeroes) can be described by the regular expression $\{0\} \cup (\{1\} \cdot \{0, 1\}^*)$. Alternatively, one could describe these numbers also in a recursive way as the following example shows.

Example 1.5. If one wants to write down a binary number, one has the following recursive rules:

- A binary number can just be the string “0”;
- A binary number can be a string “1” followed by some digits;
- Some digits can either be “0” followed by some digits or “1” followed by some digits or just the empty string.

So the binary number 101 consists of a 1 followed by some digits. These some digits consists of a 0 followed by some digits; now these some digits can again be described as a 1 followed by some digits; the remaining some digits are now void, so one can describe them by the empty string and the process is completed. Formally, one can use S to describe binary numbers and T to describe some digits and put the rules into this form:

- $S \rightarrow 0$;
- $S \rightarrow 1T$;
- $T \rightarrow T0$, $T \rightarrow T1$, $T \rightarrow \varepsilon$.

Now the process of making 101 is obtained by applying the rules iteratively: $S \rightarrow 1T$ to S giving $1T$; now $T \rightarrow 0T$ to the T in $1T$ giving $10T$; now $T \rightarrow 1T$ to the T in $10T$ giving $101T$; now $T \rightarrow \varepsilon$ to the T in $101T$ giving 101. Such a process is described by a grammar.

Grammars have been formalised by linguists as well as by mathematicians. They trace in mathematics back to Thue [87] and in linguistics, Chomsky [17] was one of the founders. Thue mainly considered a set of strings over a finite alphabet Σ with rules of the form $l \rightarrow r$ such that every string of the form xly can be transformed into xry by applying that rule. A Thue-system is given by a finite alphabet Σ and a finite set of rules where for each rule $l \rightarrow r$ also the rule $r \rightarrow l$ exists; a semi-Thue-system does not need to permit for each rule also the inverted rule. Grammars are in principle semi-Thue-systems, but they have made the process of generating the words more formal. The main idea is that one has additional symbols, so called non-terminal symbols, which might occur in the process of generating a word but which are not permitted to be in the final word. In the introductory example, S (binary numbers) and T (some digits) are the non-terminal symbols and $0, 1$ are the terminal digits. The formal definition is the following.

Definition 1.6. *A grammar (N, Σ, P, S) consists of two disjoint finite sets of symbols N and Σ, a set of rules P and a starting symbol $S \in N$.*

Each rule is of the form $l \to r$ where l is a string containing at least one symbol from N.

v can be derived from w in one step iff there are x, y and a rule $l \to r$ such that $v = xly$ and $w = xrw$. v can be derived from w in arbitrary steps iff there are $n \geq 0$ and $u_0, u_1, \ldots, u_n \in (N \cup \Sigma)^*$ such that $u_0 = v$, $u_n = w$ and u_{m+1} can be derived from u_m in one step for each $m < n$.

Now (N, Σ, P, S) generates the set $L = \{w \in \Sigma^* : w$ can be derived from $S\}$.

Convention. One writes $v \Rightarrow w$ for saying that w can be derived from v in one step and $v \Rightarrow^* w$ for saying that w can be derived from v (in an arbitrary number of steps).

Example 1.7. Let $N = \{S, T\}$, $\Sigma = \{0, 1\}$, P contain the rules $S \to 0T1, T \to 0T, T \to T1, T \to 0, T \to 1$ and S be the start symbol.

Then $S \Rightarrow^* 001$ and $S \Rightarrow^* 011$: $S \Rightarrow 0T1 \Rightarrow 001$ and $S \Rightarrow 0T1 \Rightarrow 011$ by applying the rule $S \to 0T1$ first and then either $T \to 0$ or $T \to 1$. Furthermore, $S \Rightarrow^* 0011$ by $S \Rightarrow 0T1 \Rightarrow 0T11 \Rightarrow 0011$, that is, by applying the rules $S \to 0T1$, $T \to T1$ and $T \to 0$. $S \not\Rightarrow^* 000$ and $S \not\Rightarrow^* 111$ as the first rule must be $S \to 0T1$ and any word generated will preserve the 0 at the beginning and the 1 at the end.

This grammar generates the language of all strings which have at least 3 symbols and which consist of 0s followed by 1s where there must be at least one 0 and one 1.

Example 1.8. Let $(\{S\}, \{0, 1\}, P, S)$ be a grammar where P consists of the four rules $S \to SS|0S1|1S0|\varepsilon$.

Then $S \Rightarrow^* 0011$ by applying the rule $S \to 0S1$ twice and then applying $S \to \varepsilon$. Furthermore, $S \Rightarrow^* 010011$ which can be seen as follows: $S \Rightarrow SS \Rightarrow 0S1S \Rightarrow 01S \Rightarrow 010S1 \Rightarrow 0100S11 \Rightarrow 010011$.

This grammar generates the language of all strings in $\{0, 1\}^*$ which contain as many 0s as 1s.

Example 1.9. Let $(\{S, T\}, \{0, 1, 2\}, P, S)$ be a grammar where P consists of the rules $S \to 0T|1T|2T|0|1|2$ and $T \to 0S|1S|2S$.

Then $S \Rightarrow^* w$ iff $w \in \{0, 1, 2\}^*$ and the length of w is odd; $T \Rightarrow^* w$ iff $w \in \{0, 1, 2\}^*$

and the length of w is even but not 0.

This grammar generates the language of all strings over $\{0,1,2\}$ which have an odd length.

Exercise 1.10. *Make a grammar which generates all strings with four* 1*s followed by one* 2 *and arbitrary many* 0*s in between. That is, the grammar should correspond to the regular expression* $0^*10^*10^*10^*10^*20^*$.

The Chomsky Hierarchy. Noam Chomsky [17] studied the various types of grammars and introduced the hierarchy named after him; other pioneers of the theory of formal languages include Marcel-Paul Schützenberger. The Chomsky hierarchy has four main levels; these levels were later refined by introducing and investigating other classes of grammars and formal languages defined by them.

Definition 1.11. *Let* (N,Σ,P,S) *be a grammar. The grammar belongs to the first of the following levels of the Chomsky hierarchy which applies:*

(CH3) *The grammar is called regular (or right-linear) if every rule (member of* P*) is of the form* $A \to wB$ *or* $A \to w$ *where* A,B *are non-terminals and* $w \in \Sigma^*$*. A language is regular iff it is generated by a regular grammar.*

(CH2) *The grammar is called context-free iff every rule is of the form* $A \to w$ *with* $A \in N$ *and* $w \in (N \cup \Sigma)^*$*. A language is context-free iff it is generated by a context-free grammar.*

(CH1) *The grammar is called context-sensitive iff every rule is of the form* $uAw \to uvw$ *with* $A \in N$ *and* $u,v,w \in (N \cup \Sigma)^*$ *and* $v \neq \varepsilon$*; furthermore, in the case that the start symbol* S *does not appear on any right side of a rule, the rule* $S \to \varepsilon$ *can be added so that the empty word can be generated. A language is called context-sensitive iff it is generated by a context-sensitive grammar.*

(CH0) *There is the most general case where the grammar does not satisfy any of the three restrictions above. A language is called recursively enumerable iff it is generated by some grammar.*

The next theorem permits easier methods to prove that a language is context-sensitive by constructing the corresponding grammars.

Theorem 1.12. *A language* L *not containing* ε *is context-sensitive iff it can be generated by a grammar* (N,Σ,P,S) *satisfying that every rule* $l \to r$ *satisfies* $|l| \leq |r|$*.*

A language L *containing* ε *is context-sensitive iff it can be generated by a grammar*

(N,Σ,P,S) satisfying that $S \to \varepsilon$ is a rule and that any further rule $l \to r$ satisfies $|l| \leq |r| \wedge r \in (N \cup \Sigma - \{S\})^*$.

Example 1.13. The grammar $(\{S,T,U\},\{0,1,2\},P,S)$ with P consisting of the rules $S \to 0T12|012|\varepsilon$, $T \to 0T1U|01U$, $U1 \to 1U$, $U2 \to 22$ generates the language of all strings $0^n1^n2^n$ where n is a natural number (including 0).

For example, $S \Rightarrow 0T12 \Rightarrow 00T1U12 \Rightarrow 00T11U2 \Rightarrow 00T1122 \Rightarrow 0001U1122 \Rightarrow 00011U122 \Rightarrow 000111U22 \Rightarrow 000111222$.

One can also see that the numbers of the 0s, 1s and 2s generated are always the same: the rules $S \to 0T12$ and $S \to 012$ and $S \to \varepsilon$ produce the same quantity of these symbols; the rules $T \to 0T1U$ and $T \to 01U$ produce one 0, one 1 and one U which can only be converted into a 2 using the rule $U2 \to 22$ but cannot be converted into anything else; it must first move over all 1s using the rule $U1 \to 1U$ in order to meet a 2 which permits to apply $U2 \to 22$. Furthermore, one can see that the resulting string has always the 0s first, followed by 1s and the 2s last. Hence every string generated is of the form $0^n1^n2^n$.

Note that the notion of regular language is the same whether it is defined by a regular grammar or by a regular expression.

Theorem 1.14. *A language L is generated by a regular expression iff it is generated by a regular grammar.*

Proof. One shows by induction that every language generated by a regular expression is also generated by a regular grammar. A finite language $\{w_1,w_2,\ldots,w_n\}$ is generated by the grammar with the rules $S \to w_1|w_2|\ldots|w_n$. For the inductive sets, assume now that L and H are regular sets (given by regular expressions) which are generated by the grammars (N_1,Σ,P_1,S_1) and (N_2,Σ,P_2,S_2), where the sets of non-terminals are disjoint: $N_1 \cap N_2 = \emptyset$. Now one can make a grammar $(N_1 \cup N_2 \cup \{S,T\},\Sigma,P,S)$ where P depends on the respective case of $L \cup H$, $L \cdot H$ and L^*. The set P of rules (with A,B being non-terminals and w being a word of terminals) is defined as follows in the respective case:

Union $L \cup H$: P contains all rules from $P_1 \cup P_2$ plus $S \to S_1|S_2$;

Concatenation $L \cdot H$: P contains the rules $S \to S_1$, $T \to S_2$ plus all rules of the form $A \to wB$ which are in $P_1 \cup P_2$ plus all rules of the form $A \to wT$ with $A \to w$ in P_1 plus all rules of the form $A \to w$ in P_2;

Kleene Star L^*: P contains the rules $S \to S_1$ and $S \to \varepsilon$ and each rule $A \to wB$ which is in P_1 and each rule $A \to wS$ for which $A \to w$ is in P_1.

It is easy to see that in the case of the union, a word w can be generated iff one uses the rule $S \to S_1$ and $S_1 \Rightarrow^* w$ or one uses the rule $S \to S_2$ and $S_2 \Rightarrow^* w$. Thus $S \Rightarrow^* w$ iff $w \in L$ or $w \in H$.

In the case of a concatenation, a word u can be generated iff there are v, w such that $S \Rightarrow^* S_1 \Rightarrow^* vT \Rightarrow vS_2 \Rightarrow^* vw$ and $u = vw$. This is the case iff L contains v and H contains w: $S_1 \Rightarrow^* vT$ iff one can, by same rules with only the last one changed to have the final T omitted derive that $v \in L$ for the corresponding grammar; $T \Rightarrow^* w$ iff one can derive in the grammar for H that $w \in L$. Here T was introduced for being able to give this formula; one cannot use S_2 directly as the grammar for H might permit that $S_2 \Rightarrow^* tS_2$ for some non-empty word t.

The ingredient for the verification of the grammar for Kleene star is that $S_1 \to uS$ without using the rule $S \to S_1$ iff $S_1 \to u$ can be derived in the original grammar for L; now one sees that $S \to^* uS$ for non-empty words in the new grammar is only possible iff $u = u_1 u_2 \ldots u_n$ for some n and words $u_1, u_2, \ldots, u_n \in L$; furthermore, the empty word can be generated.

For the converse direction, assume that a regular grammar with rules $R_1, R_2, \ldots, R_n$ is given. One makes a sequence of regular expressions $E_{C,D,m}$ and $E_{C,m}$ where C, D are any non-terminals and which will satisfy the following conditions:

- $E_{C,D,m}$ generates the language of words v for which there is a derivation $C \Rightarrow^* vD$ using only the rules $R_1, R_2, \ldots, R_m$;
- $E_{C,m}$ generates the language of all words v for which there is a derivation $C \Rightarrow^* v$ using only the rules $R_1, R_2, \ldots, R_m$.

One initialises all $E_{C,0} = \emptyset$ and if $C = D$ then $E_{C,D} = \{\varepsilon\}$ else $E_{C,D} = \emptyset$. If $E_{C,m}$ and $E_{C,D,m}$ are defined for $m < n$, then one defines the expressions $E_{C,m+1}$ and $E_{C,D,m+1}$ in dependence of what R_{m+1} is.

If R_{m+1} is of the form $A \to w$ for a non-terminal A and a terminal word w then one defines the updated sets as follows for all C, D:

- $E_{C,D,m+1} = E_{C,D,m}$, as one cannot derive anything ending with D with help of R_{m+1} what can not already be derived without help of R_{m+1};
- $E_{C,m+1} = E_{C,m} \cup (E_{C,A,m} \cdot \{w\})$, as one can either only use old rules what is captured by $E_{C,m}$ or go from C to A using the old rules and then terminating the derivation with the rule $A \to w$.

In both cases, the new expression is used by employing unions and concatenations and thus is in both cases again a regular expression.

If R_{m+1} is of the form $A \to wB$ for non-terminals A, B and a terminal word w then one defines the updated sets as follows for all C, D:

- $E_{C,D,m+1} = E_{C,D,m} \cup E_{C,A,m} \cdot w \cdot (E_{B,A,m} \cdot w)^* \cdot E_{B,D,m}$, as one can either directly go from C to D using the old rules or go to A employing the rule and producing a w and then ending up in B with a possible repetition by going be to A and employing again the rule making a w finitely often and then go from B to D;
- $E_{C,m+1} = E_{C,m} \cup E_{C,A,m} \cdot w \cdot (E_{B,A,m} \cdot w)^* \cdot E_{B,m}$, as one can either directly generate a terminal word using the old rules or go to A employing the rule and producing a w and then ending up in B with a possible repetition by going be to A and employing again the rule making a w finitely often and then employ more rules to finalise the making of the word.

Again, the new regular expressions put together the old ones using union, concatenation and Kleene star only. Thus one obtains also on level $m + 1$ a set of regular expressions.

After one has done this by induction for all the rules in the grammar, the resulting expression $E_{S,n}$ where S is the start symbol generates the same language as the given grammar did. This completes the second part of the proof. ▮

For small examples, one can write down the languages in a more direct manner, though it is still systematic.

Example 1.15. Let L be the language $(\{0,1\}^* \cdot 2 \cdot \{0,1\}^* \cdot 2) \cup \{0,2\}^* \cup \{1,2\}^*$.

A regular grammar generating this language is $(\{S,T,U,V,W\}, \{0,1,2\}, P, S)$ with the rules $S \rightarrow T|V|W$, $T \rightarrow 0T|1T|2U$, $U \rightarrow 0U|1U|2$, $V \rightarrow 0V|2V|\varepsilon$ and $W \rightarrow 1W|2W|\varepsilon$.

Using the terminology of Example 1.17, $L_U = \{0,1\}^* \cdot 2$, $L_T = \{0,1\}^* \cdot 2 \cdot L_U = \{0,1\}^* \cdot 2 \cdot \{0,1\}^* \cdot 2$, $L_V = \{0,2\}^*$, $L_W = \{1,2\}^*$ and $L = L_S = L_T \cup L_V \cup L_W$.

Exercise 1.16. *Let L be the language $(\{00,11,22\} \cdot \{33\}^*)^*$. Make a regular grammar generating the language.*

Example 1.17. Let $(\{S,T\}, \{0,1,2,3\}, P, S)$ be a given regular grammar.

For $A, B \in \{S,T\}$, let $L_{A,B}$ be the finite set of all words $w \in \{0,1,2,3\}^*$ such that the rule $A \rightarrow wB$ exists in P and let L_A be the finite set of all words $w \in \{0,1,2,3\}^*$ such that the rule $A \rightarrow w$ exists in P. Now the grammar generates the language

$$(L_{S,S})^* \cdot (L_{S,T} \cdot (L_{T,T})^* \cdot L_{T,S} \cdot (L_{S,S})^*)^* \cdot (L_S \cup L_{S,T} \cdot (L_{T,T})^* \cdot L_T).$$

For example, if P contains the rules $S \rightarrow 0S|1T|2$ and $T \rightarrow 0T|1S|3$ then the language generated is

$$0^* \cdot (10^*10^*)^* \cdot (2 \cup 10^*3)$$

which consists of all words from $\{0,1\}^* \cdot \{2,3\}$ such that either the number of 1s is even and the word ends with 2 or the number of 1s is odd and the word ends with 3.

Exercise 1.18. *Let $(\{S,T,U\},\{0,1,2,3,4\},P,S)$ be a grammar where the set P contains the rules $S \to 0S|1T|2$, $T \to 0T|1U|3$ and $U \to 0U|1S|4$. Make a regular expression describing this language.*

The Pumping Lemmas are methods to show that certain languages are not regular or not context-free. These criteria are only sufficient to show that a language is more complicated than assumed, they are not necessary. The following version is the standard version of the pumping lemma.

Theorem 1.19: Pumping Lemma. **(a)** *Let $L \subseteq \Sigma^*$ be an infinite regular language. Then there is a constant k such that for every $u \in L$ of length at least k there is a representation $x \cdot y \cdot z = u$ such that $|xy| \leq k$, $y \neq \varepsilon$ and $xy^*z \subseteq L$.*

(b) *Let $L \subseteq \Sigma^*$ be an infinite context-free language. Then there is a constant k such that for every $u \in L$ of length at least k there is a representation $vwxyz = u$ such that $|wxy| \leq k$, $w \neq \varepsilon \vee y \neq \varepsilon$ and $vw^\ell xy^\ell z \in L$ for all $\ell \in \mathbb{N}$.*

Comment. Pumping Lemmas like this one can also be seen as a game. They come with a pumping conditions and for a language L, there are two players. The first player Anke wants to show that L satisfies the respective pumping condition and the second player Boris wants to show that this is not the case. One player of the two has a winning strategy, that is, the player always wins then the player does the right moves. If Anke has a winning strategy then L satisfies the pumping condition; if Boris has a winning strategy then L does not satisfy the pumping condition. The game is the following (with options (a) and (b) in step 3 referring to the respective parts of the puming lemma).

1. Anke selects a pumping constant k;
2. If there are words in L of length at least k then Boris selects such a word z else Anke has won the game;
3. Anke splits the word into parts u, v, w, x, y such that $z = uvwxy$ and $|v| \geq 1$ and for (a) $|uv| \leq k$ and $x = y = \varepsilon$ and for (b) $|vwx| \leq k$;
4. Boris selects natural number $h \in \{0,1,2,\ldots\}$;
5. If $uv^hwx^hy \in L$ then Anke wins the game else Boris wins the game.

So L satisfies the pumping condition iff Anke has a winning strategy; this strategy only needs to exist, it does not need to be effective in any way; however, for regular and context-free languages, the winning strategies can be computed from the respective regular expression or grammar in cases (a) and (b), respectively.

Proof. Part (a): One considers for this proof only regular expressions constructed by finite sets and unions, concatenations and Kleene star of other expressions. For regular expressions σ, let $L(\sigma)$ be the language described by σ. Now assume that σ is a shortest regular expression such that for $L(\sigma)$ fails to satisfy the Pumping Lemma. One of the following cases must apply to σ:

First, $L(\sigma)$ is a finite set given by an explicit list in σ. Let k be a constant longer than every word in $L(\sigma)$. Then the Pumping Lemma would be satisfied as it only requests any condition on words in L which are longer than k – there are no such words.

Second, σ is $(\tau \cup \rho)$ for further regular expressions τ, ρ. As τ, ρ are shorter than σ, $L(\tau)$ satisfies the Pumping Lemma with constant k' and $L(\rho)$ with constant k''; let $k = \max\{k', k''\}$. Consider any word $w \in L(\sigma)$ which is longer than k. If $w \in L(\tau)$ then $|w| > k'$ and $w = xyz$ for some x, y, z with $y \neq \varepsilon$ and $|xy| \leq k'$ and $xy^*z \subseteq L(\tau)$. It follows that $|xy| \leq k$ and $xy^*z \subseteq L(\sigma)$. Similarly, if $w \in L(\rho)$ then $|w| > k''$ and $w = xyz$ for some x, y, z with $y \neq \varepsilon$ and $|xy| \leq k''$ and $xy^*z \subseteq L(\rho)$. It again follows that $|xy| \leq k$ and $xy^*z \subseteq L(\sigma)$. Thus the Pumping Lemma also holds in this case with the constant $k = \max\{k', k''\}$.

Third, σ is $(\tau \cdot \rho)$ for further regular expressions τ, ρ. As τ, ρ are shorter than σ, $L(\tau)$ satisfies the Pumping Lemma with constant k' and $L(\rho)$ with constant k''; let $k = k' + k''$. Consider any word $u \in L(\sigma)$ which is longer than k. Now $u = vw$ with $v \in L(\tau)$ and $w \in L(\rho)$. If $|v| > k'$ then $v = xyz$ with $y \neq \varepsilon$ and $|xy| \leq k'$ and $xy^*z \subseteq L(\tau)$. It follows that $|xy| \leq k$ and $xy^*(zw) \subseteq L(\sigma)$, so the Pumping Lemma is satisfied with constant k in the case $|v| > k'$. If $|v| \leq k'$ then $w = xyz$ with $y \neq \varepsilon$ and $|xy| \leq k''$ and $xy^*z \subseteq L(\rho)$. It follows that $|(vx)y| \leq k$ and $(vx)y^*z \subseteq L(\sigma)$, so the Pumping Lemma is satisfied with constant k in the case $|v| \leq k'$ as well.

Fourth, σ is τ^* for further regular expression τ. Then τ is shorter than σ and $L(\tau)$ satisfies the Pumping Lemma with some constant k. Now it is shown that $L(\sigma)$ satisfies the Pumping Lemma with the same constant k. Assume that $v \in L(\sigma)$ and $|v| > k$. Then $v = w_1 w_2 \ldots w_n$ for some $n \geq 1$ and non-empty words $w_1, w_2, \ldots, w_n \in L(\tau)$. If $|w_1| \leq k$ then let $x = \varepsilon$, $y = w_1$ and $z = w_2 \cdot \ldots \cdot w_n$. Now $xy^*z = w_1^* w_2 \ldots w_n \subseteq L(\tau)^* = L(\sigma)$. If $|w_1| > k$ then there are x, y, z with $w_1 = xyz$, $|xy| \leq k$, $y \neq \varepsilon$ and $xy^*z \subseteq L(\tau)$. It follows that $xy^*(z \cdot w_2 \cdot \ldots \cdot w_n) \subseteq L(\sigma)$. Again the Pumping Lemma is satisfied.

It follows from this case distinction that the Pumping Lemma is satisfied in all cases and therefore the regular expression σ cannot be exist as assumed. Thus all regular languages satisfy the Pumping Lemma.

Part (b) is omitted; see the lecture notes on Theory of Computation. ∎

In Section 2 below a more powerful version of the pumping lemma for regular sets will be shown. The following weaker corollary might also be sufficient in some cases to show that a language is not regular.

Corollary 1.20. *Assume that L is an infinite regular language. Then there is a constant k such that for each word $w \in L$ with $|w| > k$, one can represent w as $xyz = w$ with $y \neq \varepsilon$ and $xy^*z \subseteq L$.*

Exercise 1.21. *Let $p_1, p_2, p_3, \ldots$ be the list of prime numbers in ascending order. Show that $L = \{0^n : n > 0$ and $n \neq p_1 \cdot p_2 \cdot \ldots \cdot p_m$ for all $m\}$ satisfies Corollary 1.20 but does not satisfy Theorem 1.19 (a).*

Exercise 1.22. *Assume that (N, Σ, P, S) is a regular grammar and h is a constant such that N has less than h elements and for all rules of the form $A \to wB$ or $A \to w$ with $A, B \in N$ and $w \in \Sigma^*$ it holds that $|w| < h$. Show that Theorem 1.19 (a) holds with the constant k being h^2.*

Exercise 1.23. *Prove a weaker version of Theorem 1.19 (b) without requesting that the $|wxy| \leq k$. The idea to be used is that there is a constant k such that for every word $u \in L$ which is longer than k, one can be split into $vwxyz$ such that there is a non-terminal A with $S \Rightarrow^* vAz \Rightarrow^* vwAyz \Rightarrow^* vwxyz$. Then $A \Rightarrow^* wAy$ is equivalent to $vAz \Rightarrow^* vwAyz$ and use this fact to derive the pumping lemma.*

Example 1.24. The set $L = \{0^p : p$ is a prime number$\}$ of all 0-strings of prime length is not context-free.

To see this, assume the contrary and assume that k is the constant from the pumping condition in Theorem 1.19 (b). Let p be a prime number larger than k. Then 0^p can be written in the form $vwxyz$ with $q = |wy| > 0$. Then every string of the form $vw^\ell xy^\ell z$ is in L; these strings are of the form $0^{p+q\cdot(\ell-1)}$. Now choose $\ell = p+1$ and consider $0^{p+q\cdot p}$. The number $p + q \cdot p = p \cdot (q+1)$ is not a prime number; however $0^{p+q\cdot p}$ is in L by the pumping condition in Theorem 1.19 (b). This contradiction proves that L cannot be context-free.

Example 1.25. The language L of all words which have as many 0 as 1 satisfies the pumping condition in Corollary 1.20 but not the pumping condition in Theorem 1.19 (a).

For seeing the first, note that whenever w has as many 0 as 1 then every element of w^* has the same property. Indeed, $L = L^*$ and Corollary 1.20 is satisfied by every language which is of the form H^* for some H.

For seeing the second, assume the contrary and assume that n is the constant used in Theorem 1.19 (a). Now consider the word 0^n1^n. By assumption there is a representation $xyz = 0^n1^n$ with $|xy| \leq n$ and $y \neq \varepsilon$. As a consequence, $xyyz = 0^{n+m}1^n$ for some $m > 0$ and $xyyz \notin L$. Hence the statement in Theorem 1.19 (a) is not satisfied.

Theorem 1.26. *Let $L \subseteq \{0\}^*$. The following conditions are equivalent for L:*

(a) *L is regular;*
(b) *L is context-free;*
(c) *L satisfies the Theorem 1.19 (a) for regular languages;*
(d) *L satisfies the Theorem 1.19 (b) for context-free languages.*

Proof. Clearly (a) implies (b),(c) and (b),(c) both imply (d). Now it will be shown that (d) implies (a).

Assume that k is the pumping constant for the context-free Pumping Lemma. Then, for every word $u \in L$, one can split 0^n into $vwxyz$ such that $|wxy| \leq k$ and at least one of w, y is not empty and $vw^hxy^hz \in L$ for all h.

Now when $h - 1 = \ell \cdot k!/|wy|$ for some integer ℓ, the word vw^hxy^hz is equal to $0^n \cdot 0^{k!\cdot\ell}$. As all these vw^hxy^hz are in L, it follows that $0^n \cdot (0^{k!})^* \subseteq L$. For each remainder $m \in \{0, 1, \ldots, k! - 1\}$, let

$$n_m = \min\{i : \exists j\, [i > k \textit{ and } i = m + jk! \textit{ and } 0^i \in L]\}$$

and let $n_m = \infty$ when there is no such i, that is, $\min \emptyset = \infty$.

Now L is the union of finitely many regular sets: First the set $L \cap \{\varepsilon, 0, 00, \ldots, 0^k\}$ which is finite and thus regular; Second, all those sets $0^{n_m} \cdot (0^{k!})^*$ where $m < k!$ and $n_m < \infty$. There are at most $k!$ many of these sets of the second type and each is given by a regular expression. Thus L is the union of finitely many regular sets and therefore regular itself. ▮

Exercise 1.27. *Consider the following languages:*

- $L = \{0^n1^n2^n : n \in \mathbb{N}\}$;
- $H = \{0^n1^m : n^2 \leq m \leq 2n^2\}$;
- $K = \{0^n1^m2^k : n \cdot m = k\}$.

Show that these languages are not context-free using Theorem 1.19 (b).

Exercise 1.28. *Construct a context-sensitive grammar for $\{10^n1 : n$ is a power of three$\}$. Here the powers of three are $1, 3, 9, 27, \ldots$ and include the zeroth power of three.*

Exercise 1.29. *Construct a context-sensitive grammar for $\{10^n1 : n$ is at least four and not a prime$\}$.*

Exercise 1.30. *Construct a context-free grammar for the language $\{uvw \in \{0,1\}^*$: $|u| = |v| = |w|$ and $u \neq w$.*

Exercise 1.31. *Let $F(L) = \{v : \exists w \in L\, [v$ is obtained by reordering the symbols in $w]\}$. Reorderings include the void reordering where all digits remain at their position. So $F(\{0, 00, 01\}) = \{0, 00, 01, 10\}$. Determine the possible levels in Chomsky hierarchy which $F(L)$ can have when L is regular. For each possible level, exhibit a regular language L such that $F(L)$ is exactly on that level.*

Exercise 1.32. *Let $F(L)$ as in Exercise 1.31 and consider the following weaker version of Theorem 1.19 (b): There is a constant c such that all words in $u \in F(L)$ with $|u| \geq c$ can be represented as $u = v \cdot w \cdot x \cdot y \cdot z$ with $w \cdot y \neq \varepsilon$ and $vw^n xy^n z \in F(L)$ for all $n \in \mathbb{N}$. Provide a regular language L such that $F(L)$ satisfies this weaker version of the pumping lemma but neither Theorem 1.19 (b) nor Corollary 1.20.*

Exercise 1.33. *Let $L = \{0^n 1^m 2^k : n \neq m \vee n \neq k \vee m \neq k\}$. As L is context-free, it satisfies all pumping lemmas satisfied by context-free languages. Show that L satisfies also Corollary 1.20 with the additional constraint that both the constant and the length of the pump is 1, that is the following holds: $(*)$ There is a constant k such that every word $w \in L$ of length at least $k+1$ can be split into $xyz = w$ with $|y| = 1$ such that $xy^*z \subseteq L$. Show that $F(L)$ also satisfies $(*)$. Furthermore, is it true that whenever a language H satisfies $(*)$ so does $F(H)$? Here $F(L)$ and $F(H)$ are defined as in Exercise 1.31.*

Exercise 1.34. *For given L, let $G(L) = \{vw : wv \in L$ and $v, w \in \Sigma^*\}$ and note that $L \subseteq G(L)$, as it can be that v or w is ε in the above formula for $G(L)$. Provide all levels of the Chomsky hierarchy for which there is an L exactly on this level such that $G(L)$ is regular; note that when the membership problem of a language L cannot be solved by an algorithm in exponential time then L is not context-sensitive.*

Exercise 1.35. *Let $L = \{w \in \{0, 1, 2, 3\}^* :$ if $a < b$ then b occurs more frequently than $a\}$. What is the exact level of L in the Chomsky hierarchy? Use grammars and pumping lemmas to prove the result.*

Exercise 1.36. *Let L be given by the grammar $(\{S\}, \{0, 1\}, \{S \to 01S|01, S0 \to 0S, S1 \to 1S, 0S \to S0, 1S \to S1\}, S)$. Determine the level of L in the Chomsky hierarchy, it is one of regular, context-free and context-sensitive, as it is given by a context-sensitive grammar. Determine all words up to length 6 in L and explain verbally when a word belongs to L.*

Exercise 1.37. *Construct context-free grammars for the sets $L = \{0^n 1^m 2^k : n < m \vee m < k\}$, $H = \{0^n 1^m 2^{n+m} : n, m \in \mathbb{N}\}$ and $K = \{w \in \{0, 1, 2\}^* : w$ has a subword of the form $20^n 1^n 2$ for some $n > 0$ or $w = \varepsilon\}$.*

Which of the versions of the Pumping Lemma (Theorems 1.19 (a) and 1.19 (b) and Corollary 1.20) are satisfied by L, H and K, respectively.

Exercise 1.38. *Let* $L = \{0^h 1^i 2^j 3^k : (h \neq i \text{ and } j \neq k) \text{ or } (h \neq k \text{ and } i \neq j)\}$ *be given. Construct a context-free grammar for L and determine which of versions of the Pumping Lemma (Theorems 1.19 (a) and 1.19 (b) and Corollary 1.20) are satisfied by L.*

Exercise 1.39. *Consider the grammar* $(\{S\}, \{0, 1, 2, 3\}, \{S \to 00S|S1|S2|3\}, S)$ *and construct for the language L generated by the grammar the following: a regular grammar for L and a regular expression for L.*

In the following exercises, let $f_L(n)$ be the number of words $w \in L$ with $|w| < n$. So if $L = \{0\}^*$ then $f_L(n) = n$ and if $L = \{0, 1\}^*$ then $f_L(n) = 2^n - 1$.

Exercise 1.40. *Is there a context-free language L with* $f_L(n) = \lfloor \sqrt{n} \rfloor$*, where* $\lfloor \sqrt{n} \rfloor$ *is the largest integer bounded by* $\sqrt{n}$*? Either prove that there is no such set or construct a set with the corresponding context-free grammar.*

Exercise 1.41. *Is there a regular set L with* $f_L(n) = n(n + 1)/2$*? Either prove that there is no such set or construct a set with the corresponding regular grammar or regular expression.*

Exercise 1.42. *Is there a context-sensitive set L with* $f_L(n) = n^n$*, where* $0^0 = 0$*? Either prove that there is no such set or construct a set with the corresponding grammar.*

Exercise 1.43. *Is there a regular set L with* $f_L(n) = (3^n - 1)/2 + \lfloor n/2 \rfloor$*? Either prove that there is no such set or construct a set with the corresponding regular grammar or regular expression.*

Exercise 1.44. *Is there a regular set L with* $f_L(n) = \lfloor n/3 \rfloor + \lfloor n/2 \rfloor$*? Either prove that there is no such set or construct a set with the corresponding regular grammar or regular expression.*

For the following exercises, call y a pump of $xyz \in L$ iff $|y| \geq 1$ and $\{x\} \cdot \{y\}^* \cdot \{z\} \subseteq L$. For infinite languages L the optimal pump length k as witnessed by h is the smallest number k for which there is a h such that all $w \in L$ with $|w| \geq h$ have a pump of length up to k.

Exercise 1.45. *Determine the optimal pump length k and the witness length h for the language* $\{000, 111, 222\}^* \cap \{0000, 1111, 2222\}^* \cap \{00000, 11111, 22222\}^*$ *and explain the solution.*

Exercise 1.46. *Determine the optimal pump length k and the witness length h for the language of all words where the length modulo* 10 *is in* $\{1, 3, 7, 9\}$ *and explain the solution.*

Exercise 1.47. *Determine the optimal pump length k and the witness length h for the language of all words where the length modulo* 10 *is in* $\{0, 2, 4, 5, 6, 8\}$ *and explain the solution.*

Exercise 1.48. *Determine the optimal pump length k and the witness length h for the language* $\{001100110011\} \cdot \{222\}^* \cup \{0011\} \cdot \{2222\}^* \cup \{001100110011001100110011\}$ *and explain the solution.*

Exercise 1.49. *Determine the optimal pump length k and the witness length h for the language of all decimal numbers without leading zeroes which are multiples of* 512 *and explain the solution.*

In the following exercises, the task is the following: Given a set H which is interpreted as a set of decimal numbers, find an infinite set $L \subseteq H$ having the property stated in the exercise. Furthermore, if possible L should be regular and a regular expression or grammar should witness this; if L cannot be taken to be regular, but can be taken to be context-free then a context-free grammar should witness that L is context-free and one should use the pumping lemma to show that L cannot be taken regular; if L is context-sensitive then a grammar should witness this and one should use the context-free pumping lemma to prove that no infinite context-free L can solve the task.

Exercise 1.50. *Find infinite* $L \subseteq H$ *for* $H = \{10^n20^m1 : n \geq m \geq 1$ *and* $n + m$ *is even*$\}$ *such that all members of L are square numbers.*

Exercise 1.51. *Find infinite* $L \subseteq H$ *for* $H = \{10^n30^m30^k1 : 2n \geq m + k$ *and* 3 *divides* $n + m + k\}$ *such that all members of L are third powers (cubes).*

Exercise 1.52. *Find infinite* $L \subseteq H$ *for* $H = \{1\} \cdot \{0\}^+ \cdot \{3\} \cdot \{0\}^+ \cdot \{3\} \cdot \{0\}^+ \cdot \{1\} \cdot \{0\}^+$ *such that all members of L are third powers (cubes).*

2 Finite Automata

An automaton is in general a mechanism which checks whether a word is in a given language. An automaton has a number of states which memorise some information. Here an example.

Example 2.1: Divisibility by 3. Let $a_0a_1\ldots a_n$ be a decimal number. One can check whether $a_0a_1\ldots a_n$ is a multiple of 3 by the following algorithm using a memory $s \in \{0,1,2\}$ and processing in step m the digit a_m. The memory s is updated accordingly.

Case s=0 : If $a_m \in \{0,3,6,9\}$ then update $s = 0$;
if $a_m \in \{1,4,7\}$ then update $s = 1$;
if $a_m \in \{2,5,8\}$ then update $s = 2$.

Case s=1 : If $a_m \in \{0,3,6,9\}$ then update $s = 1$;
if $a_m \in \{1,4,7\}$ then update $s = 2$;
if $a_m \in \{2,5,8\}$ then update $s = 0$.

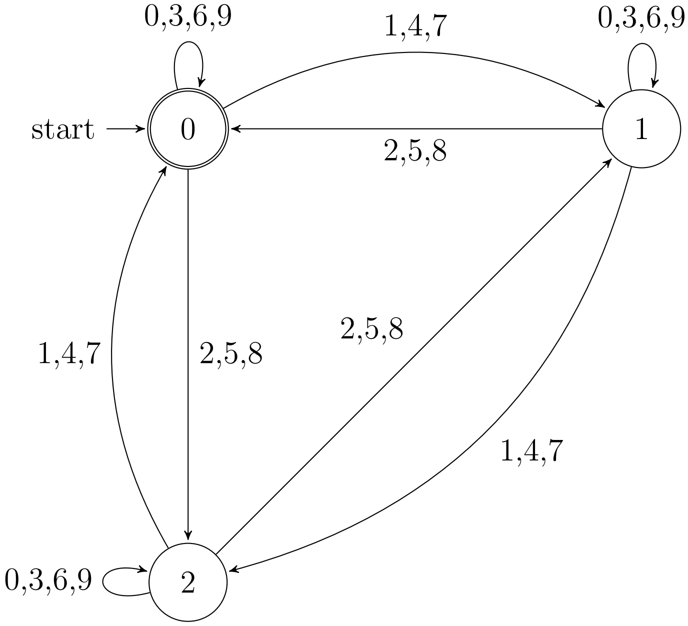
Case s=2 : If $a_m \in \{0,3,6,9\}$ then update $s = 2$;
if $a_m \in \{1,4,7\}$ then update $s = 0$;
if $a_m \in \{2,5,8\}$ then update $s = 1$.

The number $a_0a_1\ldots a_n$ is divisible by 3 iff $s = 0$ after processing a_n. For example, 123456 is divisible by 3 as the value of s from the start up to processing the corresponding digits is $0,1,0,0,1,0,0$, respectively. The number 256 is not divisible by 3 and the value of s is $0,2,1,1$ after processing the corresponding digits.

Quiz 2.2. *Which of the following numbers are divisible by* 3*:* 1*,* 20*,* 304*,* 2913*,* 49121*,* 391213*,* 2342342*,* 123454321*?*

Description 2.3: Deterministic Finite Automaton. The idea of this algorithm is to update a memory which takes only finitely many values in each step according to the digit read. At the end, it only depends on the memory whether the number which has been processed is a multiple of 3 or not. This is a quite general algorithmic method and it has been formalised in the notion of a finite automaton; for this, the possible values of the memory are called states. The starting state is the initial value of the memory. Furthermore, after processing the word it depends on the memory whether the word is in L or not; those values of the memory which say $a_0a_1\ldots a_n \in L$ are called "accepting states" and the others are called "rejecting states".

One can display the automata as a graph. The nodes of the graph are the states (possible values of the memory). The accepting states are marked with a double border, the rejecting states with a normal border. The indicator "start" or an incoming arrow mark the initial state. Arrows are labelled with those symbols on which a transition from one state to anothers takes place. Here the graphical representation of the automaton checking whether a number is divisible by 3.



Mathematically, one can also describe a finite automaton $(Q, \Sigma, \delta, s, F)$ as follows: Q is the set of states, Σ is the alphabet used, δ is the transition function mapping pairs from $Q \times \Sigma$ to Σ, s is the starting state and F is the set of accepting states.

The transition-function $\delta : Q \times \Sigma \to Q$ defines a unique extension with domain $Q \times \Sigma^*$ as follows: $\delta(q, \varepsilon) = q$ for all $q \in Q$ and, inductively, $\delta(q, wa) = \delta(\delta(q, w), a)$ for all $q \in Q$, $w \in \Sigma^*$ and $a \in \Sigma$.

For any string $w \in \Sigma^*$, if $\delta(s, w) \in F$ then the automaton accepts w else the automaton rejects w.

Example 2.4. One can also describe an automaton by a table mainly maps down δ and furthermore says which states are accepting or rejecting. The first state listed is usually the starting state. Here a table for an automaton which checks whether a number is a multiple of 7:

q	type	$\delta(q,a)$ for $a = 0$	1	2	3	4	5	6	7	8	9
0	acc	0	1	2	3	4	5	6	0	1	2
1	rej	3	4	5	6	0	1	2	3	4	5
2	rej	6	0	1	2	3	4	5	6	0	1
3	rej	2	3	4	5	6	0	1	2	3	4
4	rej	5	6	0	1	2	3	4	5	6	0
5	rej	1	2	3	4	5	6	0	1	2	3
6	rej	4	5	6	0	1	2	3	4	5	6

This automaton checks whether a number is a multiple of 7.

On input 343 the automaton goes on symbol 3 from state 0 to state 3, then on symbol 4 from state 3 to state 2 and then on symbol 3 from state 6 to state 0. The state 0 is accepting and hence 343 is a multiple of 7 (in fact $343 = 7 * 7 * 7$).

On input 999 the state goes first from state 0 to state 2, then from state 2 to state 1, then from state 1 to state 5. The state 5 is rejecting and therefore 999 is not a multiple of 7 (in fact $999 = 7 * 142 + 5$).

Example 2.5. One can also describe a finite automaton as an update function which maps finite states plus symbols to finite states by some algorithm written in a more compact form. In general the algorithm has variables taking its values from finitely many possibilities and it can read symbols until the input is exhausted. It does not have arrays or variables which go beyond its finite range. It has explicit commands to accept or reject the input. When it does "accept" or "reject" the program terminates.

```
function div257
  begin var a in {0,1,2,...,256};
        var b in {0,1,2,3,4,5,6,7,8,9};
        if exhausted(input) then reject;
        read(b,input); a = b;
        if b == 0 then
          begin if exhausted(input) then accept else reject end;
        while not exhausted(input) do
          begin read(b,input); a = (a*10+b) mod 257 end;
        if a == 0 then accept else reject end.
```

This automaton checks whether a number on the input is a multiple of 257; furthermore, it does not accept any input having leading 0s. Here some sample runs of the algorithm.

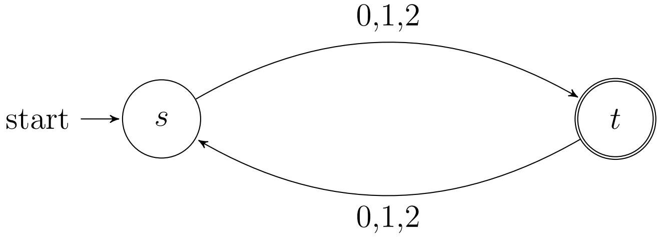
On input ε the algorithm rejects after the first test whether the input is exhausted. On input 00 the algorithm would read b one time and then do the line after the test whether b is 0; as the input is not yet exhausted, the algorithm rejects. On input 0 the algorithm goes the same way until but finally accepts the input as the input is exhausted after the symbol b has been read for the first time. On input 51657, the algorithm initialises a as 5 after having read b for the first time. Then it reaches the while-loop and, while reading $b = 1$, $b = 6$, $b = 5$, $b = 7$ it updates a to 51, 2, 25, 0, respectively. It accepts as the final value of a is 0. Note that the input 51657 is $201 * 257$ and therefore the algorithm is correct in this case.

Such algorithms permit to write automata with a large number of states in a more compact way then making a state diagram or a state table with hundreds of states.

Note that the number of states of the program is actually larger than 257, as not only the value of a but also the position in the program contributes to the state of the automaton represented by the program. The check "exhausted(input)" is there to check whether there are more symbols on the input to be processed or not; so the first check whether the input is exhausted is there to reject in the case that the input is the empty string. It is assumed that the input is always a string from $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^*$.

Exercise 2.6. *Such an algorithm might be written in a form nearer to a finite automaton if one gives the set of states explicitly, names the starting state and the accepting states and then only places an algorithm or mathematical description in order to describe δ (in place of a table). Implement the above function div257 using the state space $Q = \{s, z, r, q_0, q_1, \ldots, q_{256}\}$ where s is the starting state and z, q_0 are the accepting states; all other states are rejecting. Write down how the transition-function δ is defined as a function from $Q \times \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \to Q$. Give a compact definition and not a graph or table.*

Quiz 2.7. *Let $(\{s, t\}, \{0, 1, 2\}, \delta, s, \{t\})$ be a finite automaton with $\delta(s, a) = t$ and $\delta(t, a) = s$ for all $a \in \{0, 1, 2\}$. Determine the language of strings recognised by this automaton.*



Theorem 2.8: Characterising Regular Sets. *If a language L is recognised by a deterministic finite automaton then L is regular.*

Proof. Let an automaton $(Q, \Sigma, \delta, s, F)$ be given. Now one builds the regular grammar (Q, Σ, P, s) with the following rules:

- the rule $q \to ar$ is in P iff $\delta(q,a) = r$;
- the rule $q \to \varepsilon$ is in P iff $q \in F$.

So the non-terminals of the grammar are the states of the automaton and also the roles of every $q \in Q$ is in both constructs similar: For all $q, r \in Q$, it holds that $q \Rightarrow^* wr$ iff $\delta(q,w) = r$.

To see this, one proves it by induction. First consider $w = \varepsilon$. Now $q \Rightarrow^* wr$ iff $q = r$ iff $\delta(q,w) = r$. Then consider $w = va$ for some symbol a and assume that the statement is already proven for the shorter word v. Now $q \Rightarrow^* wr$ iff there is a non-terminal t with $q \Rightarrow^* vt \Rightarrow var$ iff there is a non-terminal t with $\delta(q,v) = t$ and $t \Rightarrow ar$ iff there is a non-terminal t with $\delta(q,v) = t$ and $\delta(t,a) = r$ iff $\delta(q,w) = r$.

The only way to produce a word w in the new grammar is to generate the word wq for some $q \in F$ and then to apply the rule $q \to \varepsilon$. Thus, the automaton accepts w iff $\delta(s,w) \in F$ iff there is a $q \in F$ with $s \Rightarrow^* q \wedge q \Rightarrow \varepsilon$ iff $s \Rightarrow^* w$. Hence w is accepted by the automaton iff w is generated by the corresponding grammar. ▮

The converse of this theorem will be shown later in Theorem 2.42.

There is a stronger version of the pumping lemma which directly comes out of the characterisation of regular languages by automata; it is called the "Block Pumping Lemma", as it says that when a word in a regular language is split into sufficiently many blocks then one can pump one non-empty sequence of these blocks.

Theorem 2.9: Block Pumping Lemma. *If L is a regular set then there is a constant k such that for all strings $u_0, u_1, \ldots, u_k$ with $u_0 u_1 \ldots u_k \in L$ and $u_1, \ldots, u_{k-1}$ being nonempty there are i, j with $0 < i < j \leq k$ and*

$$(u_0 u_1 \ldots u_{i-1}) \cdot (u_i u_{i+1} \ldots u_{j-1})^* \cdot (u_j u_{j+1} \ldots u_k) \subseteq L.$$

So if one splits a word in L into $k+1$ parts then one can select some parts in the middle of the word which can be pumped.

Block Pumping Lemma as a Game. Player Anke wants to prove that the block pumping lemma is satisfied and Boris wants to disprove this.

1. Anke selects Pumping constant k;
2. Boris selects $k+1$ blocks $u_0, u_1, \ldots, u_k$ with the inner blocks $u_1, u_2, \ldots, u_{k-1}$ being nonempty and the concatenation $u_0 u_1 \ldots u_k$ of all blocks being a word in L;
3. Anke selects i, j such that $u_i u_{i+1} \ldots u_j$ is a pump and i, j satisfy $0 < i \leq j < k$;
4. Boris selects a value $h \in \mathbb{N}$;
5. If $u_0 u_1 \ldots u_{i-1} (u_i u_{i+1} \ldots u_j)^h u_{j+1} \ldots u_k$ is in L then Anke wins the game else Boris wins the game.

Here it is a matter of taste whether one says “Boris selects a word in L and cuts it into $k+1$ blocks” or “Boris selects $k+1$ blocks such that their concatenation is a word in L”.

Proof. Given a regular set L, let $(Q, \Sigma, \delta, s, F)$ be the finite automaton recognising this language. Let $k = |Q|+1$ and consider any strings $u_0, u_1, \ldots, u_k$ with $u_0 u_1 \ldots u_k \in L$ where the inner strings $u_1, u_2, \ldots, u_{k-1}$ are not empty. There are i and j with $0 < i < j \leq k$ such that $\delta(s, u_0 u_1 \ldots u_{i-1}) = \delta(s, u_0 u_1 \ldots u_{j-1})$; this is due to the fact that there are $|Q|+1$ many values for i, j and so two of the states have to be equal. Let $q = \delta(s, u_0 u_1 \ldots u_{i-1})$. By assumption, $q = \delta(q, u_i u_{i+1} \ldots u_{j-1})$ and so it follows that $q = \delta(s, u_0 u_1 \ldots u_{i-1}(u_i u_{i+1} \ldots u_{j-1})^h)$ for every h. Furthermore, $\delta(q, u_j u_{j+1} \ldots u_k) \in F$ and hence $u_0 u_1 \ldots u_{i-1}(u_i u_{i+1} \ldots u_{j-1})^h u_j u_{j+1} \ldots u_k \in L$ for all h. ▮

Example 2.10. Let L be the language of all strings over $\{0, 1, 2\}$ which contains an even number of 0s. Then the pumping-condition of Theorem 2.9 is satisfied with parameter $n = 3$: Given $u_0 u_1 u_2 u_3 \in L$, there are three cases:

- u_1 contains an even number of 0s. Then removing u_1 from the word or inserting it arbitrarily often does not make the number of 0s in the word odd; hence $u_0 (u_1)^* u_2 u_3 \subseteq L$.
- u_2 contains an even number of 0s. Then $u_0 u_1 (u_2)^* u_3 \subseteq L$.
- u_1 and u_2 contain both an odd number of 0s. Then $u_1 u_2$ contains an even number of 0s and $u_0 (u_1 u_2)^* u_3 \subseteq L$.

Hence the pumping condition is satisfied for L.

Let H be the language of all words which contain a different number of 0s and 1s. Let k be any constant. Now let $u_0 = 0, u_1 = 0, \ldots, u_{k-1} = 0, u_k = 1^{k+k!}$. If the pumping condition would be satisfied for H then there are i, j with $0 < i < j \leq k$ and

$$0^i (0^{j-i})^* 0^{k-j} 1^{k+k!} \subseteq H.$$

So fix this i, j and take $h = \frac{k!}{j-i} + 1$ (which is a natural number). Now one sees that $0^i 0^{(j-i)h} 0^{k-j} 1^{k+k!} = 0^{k+k!} 1^{k+k!} \notin H$, hence the pumping condition is not satisfied.

Theorem 2.11: Ehrenfeucht, Parikh and Rozenberg [25]. *A language L is regular if and only if both L and its complement satisfy the block pumping lemma.*

Proof. The proof is based on showing the even more restrictive block cancellation property. That is, it is shown that L is regular iff there is a constant $k \geq 3$ such that the following condition holds for k:

(E_k): for all words $u_0, u_1, \ldots, u_k$, there are $i, j \in \{0, 1, \ldots, k-1\}$ with $i < j$ and $L(u_0 \ldots u_k) = L(u_0 \ldots u_i \cdot u_{j+1} \ldots u_k)$.

This says in particular, if one cuts a word into $k+1$ blocks with the zeroth and the last block possibly empty then one can find an interval of some blocks not containing the zeroth and the last such that deleting the interval from the word does not change membership in L, so if x is a member of L so is the shorter word and if x is a member of the complement of L so again is the shorter word.

On one hand, it will be shown that every regular set satisfies (E_k) for some k and on the other hand that whenever a set satisfies (E_k) for some k then it is regular. Furthermore, for each k and for each fixed alphabet, there are only finitely many sets satisfying (E_k).

That regular sets satisfy (E_k) comes directly out of analysing the states of a dfa recognising the corresponding regular language with k states and by choosing i, j such that there is the same state after reading $u_0 \ldots u_i$ and after reading $u_0 \ldots u_j$.

For the other direction, one has to choose a constant $c > k$ such that every two-colouring of pairs (i, j) from $\{0, 1, \ldots, c\}$ has a homogeneous set of size $k+1$; this constant exists by the finite version of Ramsey's Theorem of Pairs [73].

Ramsey's Theorem of pairs says the following: For each k there is a c such that if one assigns to each pair (i, j) with $i < j$ and $i, j \in \{0, 1, \ldots, c-1\}$ one of the colours white or red then there is a subset $\{h_0, h_1, \ldots, h_k\}$ of $\{0, 1, \ldots, c\}$ such that all pairs $(i, j), (i', j')$ with $i < j$ and $i' < j'$ and $i, j, i', j' \in \{h_0, h_1, \ldots, h_{k-1}\}$ have the same colour. Such a subset is called homogeneous.

Ramsey's Theorem of Pairs has been a very useful tool in proving combinatorial properties in many branches of mathematics including the block pumping lemma.

Now let H_1, H_2 be two sets which satisfy (E_k) and assume they are identical on all strings of length up to c. Now assume by way of contradiction that $H_1 \neq H_2$.

Let x be the length-lexicographically first string on which $H_1(x) \neq H_2(x)$ and let u_0 be ε, u_h be the h-th symbol of x for $h = 1, \ldots, c-1$ and u_c is the remaining part of the word x. Furthermore, for $i, j \in \{0, 1, \ldots, c\}$ with $i < j$, make a two-colouring col such that the following holds: If $H_1(u_0 u_1 \ldots u_i \cdot u_{j+1} u_{j+2} \ldots u_c) = H_1(x)$ then $col(i, j) = white$ else $col(i, j) = red$.

By Ramsey's Theorem of Pairs there are $h_0, h_1, \ldots, h_k$ on which col is homogeneous and one can consider the splitting of x into $k+1$ blocks $u_0 \ldots u_{h_0}$, $u_{h_0+1} \ldots u_{h_1}$, $\ldots$, $u_{h_{k-1}+1} \ldots u_c$. These splittings again satisfy the property (E_k) for H_1. As there must be $i, j \in \{h_0, h_1, \ldots, h_{k-1}\}$ with $i < j$ and $H_1(u_0 u_1 \ldots u_{h_i} \cdot u_{h_j+1} u_{h_j+2} \ldots u_c) = H_1(x)$, the homogeneous colour is white.

Furthermore, there must, by (E_k) for H_2, exist $i', j' \in \{h_0, h_1, \ldots, h_{k-1}\}$ with $i' < j'$ and $H_2(u_1 u_2 \ldots u_{i'} \cdot u_{j'+1} u_{j'+2} \ldots u_{c'}) = H_2(x)$. Due to homogenicity, it also holds that $H_1(u_1 u_2 \ldots u_{i'} \cdot u_{j'+1} u_{j'+2} \ldots u_{c'}) = H_1(x)$. On one hand, this gives $H_1(u_1 u_2 \ldots u_{i'} \cdot$

$u_{j'+1}u_{j'+2}\ldots u_{c'}) \neq H_2(u_1u_2\ldots u_{i'}\cdot u_{j'+1}u_{j'+2}\ldots u_{c'})$, on the other hand the choice of x gives that H_1, H_2 coincide on this string as it is shorter than x. This contradiction leads to the conclusion that H_1 and H_2 coincide on all strings whenever both satisfy (E_k) and $H_1(y) = H_2(y)$ for all strings up to length c. So, whenever two sets satisfy (E_k) and when they coincide on strings up to length c then they are equal.

Note that when L satisfies (E_k), so do also all derivatives $L_x = \{y : xy \in L\}$: If $\tilde{u}_0, \tilde{u}_1, \ldots, \tilde{u}_k$ are $k+1$ strings then one considers $x\tilde{u}_0, \tilde{u}_1, \ldots, \tilde{u}_k$ for L and selects indices $i, j \in \{0, 1, \ldots, k-1\}$ with $i < j$ such that $L(x\tilde{u}_0\tilde{u}_1\ldots\tilde{u}_k) = L(x\tilde{u}_0\ldots\tilde{u}_i\cdot\tilde{u}_{j+1}\ldots\tilde{u}_k)$. It follows that $L_x(\tilde{u}_0\tilde{u}_1\ldots\tilde{u}_k) = L_x(\tilde{u}_0\ldots\tilde{u}_i\cdot\tilde{u}_{j+1}\ldots\tilde{u}_k)$ and hence also L_x satisfies (E_k).

Each derivative L_x is determined by the values $L_x(y)$ for the y with $|y| \leq c$. So there are at most $2^{1+d+d^2+\ldots+d^c}$ many derivatives where d is the number of symbols in the alphabet; in particular there are only finitely many derivatives. The language L is regular by the Theorem of Myhill and Nerode (Theorem 2.19). ▮

However, there are non-regular languages L which satisfy the block pumping lemma. Morse as well as Thue [87] constructed an infinite binary sequence in which there is no non-empty subword of the form www. This sequence witnesses that there are cubefree strings of arbitrary length and this fact is used to construct nonregular set L satisfying the block pumping lemma.

Theorem 2.12: Sequence of Morse and Thue [87]. *Let $a_0 = 0$ and, for all n, $a_{2n} = a_n$ and $a_{2n+1} = 1 - a_n$. Then the infinite binary sequence $a_0a_1\ldots$ does not contain a subword of the form www.*

Proof. In the following, call a word a "cube" if it is not empty and of the form www for some string w.

Assume by way of contradiction that the sequence of Morse and Thue contains a cube as a subword and let www be the first such subword of the sequence. Let k be the length of w and $w_1w_2\ldots w_k$ be the symbols in w (in this order).

In the case that w has even length, then consider the first position $2n+m$ with $m \in \{0,1\}$ of www in the sequence. If $m = 0$ then $a_n = a_{2n+m}, a_{n+1} = a_{2n+2+m}, \ldots, a_{n+3k/2} = a_{2n+3k}$ else $a_n = 1-a_{2n+m+1}, a_{n+1} = 1-a_{2n+2+m+1}, \ldots, a_{n+3k/2} = 1 - a_{2n+3k+1}$. In both cases, $a_na_{n+1}\ldots a_{n+3k/2}$ is of the form vvv where v has the length $k/2$ and occurs before www. As www was chosen to be the first cube occurring in the sequence, this case does not apply and k must be odd.

For the case of an odd k and for each $h \in \{1, 2, \ldots, k-1\}$, either the first or the second occurrence of w satisfies that w_h is at a position of the form $2n$ and w_{h+1} at a position of the form $2n+1$ so that, by the construction of the sequence, $w_{h+1} = 1-w_h$. Furthermore, by the same principle applied to the position where one copy of w ends and the next starts, one has that $w_1 = 1 - w_k$. However, as w has odd length, one also has $w_1 = w_k$; for example if w has length 5 then w is either 01010 or 10101. This gives a contradiction and therefore this case does also not occur. Hence the sequence of Morse and Thue has no subword which is a cube. ▮

Theorem 2.13 [14]. *There is a block pumpable language which is not regular.*

Proof. Let L contain all words which either contain a cube or whose length is not a power of 10, so nonmembers of L have one of the lengths $1, 10, 100, 1000, \ldots$ and no other length. Now one shows that L has the block pumping constant 5. Assume that $w \in L$ and w is split into blocks $u_0, u_1, u_2, u_3, u_4, u_5$ and assume that u_1, u_2, u_3, u_4 are all non-empty, as if one of them is empty one can pump that empty block. Now it is shown that one can select one of the possible pumps u_1, u_1u_2, u_3, u_3u_4 such that when omitting or repeating an arbitrary time the selected pump in w, the so modified word is again in L. In other words, one of the following languages is a subset of L: $u_0(u_1)^*u_2u_3u_4u_5$, $u_0(u_1u_2)^*u_3u_4u_5$, $u_0u_1u_2(u_3)^*u_4u_5$ and $u_0u_1u_2(u_3u_4)^*u_5$.

First consider the case that $|u_1u_2| \leq |u_3u_4|$. In this case, $|u_0u_1u_2u_1u_2u_3u_4u_5| \leq |u_0u_3u_4u_5| \cdot 3$ and only one of the words $u_0u_3u_4u_5$, $u_0u_2u_3u_4u_5$, $u_0(u_1)^2u_2u_3u_4u_5$ and $u_0(u_1u_2)^2u_3u_4u_5$ has a length which is a power of 10. Hence one can select the pump to be either u_1 or u_1u_2 such that when the pump is omitted or doubled the resulting word does not have a length which is a power of 10 and is therefore in L. Furthermore, for both possible pumps and $h \geq 3$, the words $u_0(u_1)^hu_2u_3u_4u_5$ and $u_0(u_1u_2)^hu_3u_4u_5$ do both contain a cube and are in L. Thus, one can choose the pump such that all pumped words are in L.

Second in the case that $|u_3u_4| < |u_1u_2|$, one can do the same proof as before, only with the possible pumps being u_3 and u_3u_4, one of them works.

To see that L is not regular, note that for each power of 10 there is a word in the complement of L which consists of the corresponding first symbols of the sequence of Morse and Thue. Note that the complement of L is now infinite but cannot satisfy any pumping lemma as it contains only cubefree words. Thus the complement of L and, hence, also L itself cannot be regular. ▮

Quiz 2.14. *Which of the following languages over $\Sigma = \{0, 1, 2, 3\}$ satisfy the pumping-condition from Theorem 2.9:*
(a) $\{00, 111, 22222\}^* \cap \{11, 222, 00000\}^* \cap \{22, 000, 11111\}^*$,
(b) $\{0^i1^j2^k : i + j = k + 5555\}$,
(c) $\{0^i1^j2^k : i + j + k = 5555\}$,
(d) $\{w : w$ *contains more* 1 *than* $0\}$*?*

Exercise 2.15. *Find the optimal constants for the Block Pumping Lemma for the following languages:*

(a) *$\{w \in \{0,1,2,3,4,5,6,7,8,9\}^*$: at least one nonzero digit a occurs in w at least three times$\}$;*
(b) *$\{w \in \{0,1,2,3,4,5,6,7,8,9\}^* : |w| = 255\}$;*
(c) *$\{w \in \{0,1,2,3,4,5,6,7,8,9\}^*$: the length $|w|$ is not a multiple of $6\}$;*
Here the constant for a language L is the least n such that for all words $u_0, u_1, \ldots, u_n$ the implication

$$u_0 u_1 u_2 \ldots u_n \in L \Rightarrow \exists i, j\, [0 < i < j \leq n \text{ and } u_0 \ldots u_{i-1}(u_i \ldots u_{j-1})^* u_j \ldots u_n \subseteq L]$$

holds.

Exercise 2.16. *Find the optimal constants for the Block Pumping Lemma for the following languages:*
(a) *$\{w \in \{0,1,2,3,4,5,6,7,8,9\}^* : w$ is a multiple of $25\}$;*
(b) *$\{w \in \{0,1,2,3,4,5,6,7,8,9\}^* : w$ is not a multiple of $3\}$;*
(c) *$\{w \in \{0,1,2,3,4,5,6,7,8,9\}^* : w$ is a multiple of $400\}$.*

Exercise 2.17. *Find a regular language L so that the constant of the Block Pumping Lemma for L is 4 and for the complement of L is 4196.*

Exercise 2.18. *Give an example L of a language which satisfies Theorem 1.19 (a) (where for every $w \in L$ of length at least k there is a splitting $xyz = w$ with $|xy| \leq k$, $|y| > 0$ and $xy^*z \subseteq L$) but does not satisfy Theorem 2.9 (the Block Pumping Lemma).*

Theorem 2.19: Myhill and Nerode's Minimal DFA [67]. *Given a language L, let $L_x = \{y \in \Sigma^* : xy \in L\}$ be the derivative of L to x. The language L is regular iff the number of different derivatives L_x is finite; furthermore, for languages with exactly n derivatives, one can construct a complete dfa having n and there is no complete dfa with less than n states which recognises L.*

Proof. Let $(Q, \Sigma, \delta, s, F)$ be a deterministic finite automaton recognising L. If $\delta(s,x) = \delta(s,y)$ then for all $z \in \Sigma^*$ it holds that $z \in L_x$ iff $\delta(\delta(s,x),z) \in F$ iff $\delta(\delta(s,y),z) \in F$ iff $z \in L_y$. Hence the number of different sets of the form L_x is a lower bound for the size of the states of the dfa.

Furthermore, one can directly build the dfa by letting $Q = \{L_x : x \in \Sigma^*\}$ and define for $L_x \in Q$ and $a \in \Sigma$ that $\delta(L_x, a)$ is the set L_{xa}. The starting-state is the set L_ε and $F = \{L_x : x \in \Sigma^* \wedge \varepsilon \in L_x\}$.

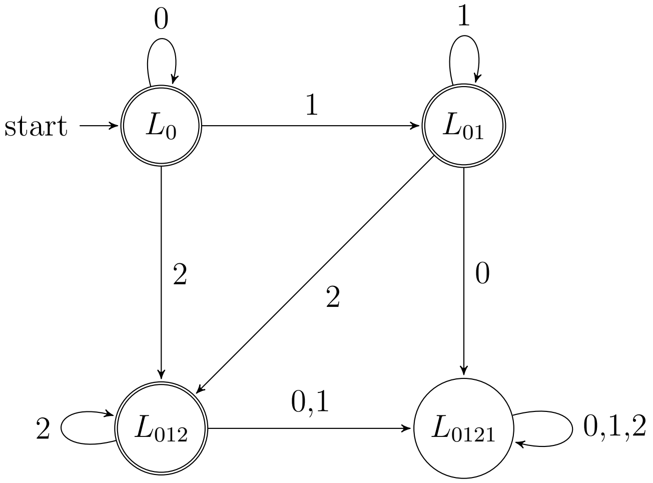
In practice, one would of course pick representatives for each state, so there is a finite subset Q of Σ^* with $\varepsilon \in Q$ and for each set L_y there is exactly one $x \in Q$ with $L_x = L_y$. Then $\delta(x,a)$ is that unique y with $L_y = L_{xa}$.

For the verification, note that there are only finitely many different derivatives, so the set Q is finite. Furthermore, each state can be reached: For $x \in Q$, one can reach the state x by feeding the word x into the automaton. Assume now that $L_x = L_y$. Then $L_{xa} = \{z : xaz \in L\} = \{z : az \in L_x\} = \{z : az \in L_y\} = \{z : yaz \in L\} = L_{ya}$, thus the transition function δ is indeed independent of whether x or y is chosen to represent L_x and will select the unique member z of Q with $L_z = L_{xa} = L_{ya}$. In addition, the rule for making exactly the states x with $\varepsilon \in L_x$ be accepting is correct: The reason is that, for $x \in Q$, the automaton is in state x after reading x and x has to be accepted by the automaton iff $x \in L$ iff $\varepsilon \in L_x$. ▮

In the case that some derivative is $\emptyset$, one can get an automaton which has one less state if one decides not to represent $\emptyset$; the resulting dfa would then be incomplete, that is, there would be nodes q and symbols a with $\delta(q, a)$ being undefined; if the automaton ends up in this situation, it would just reject the input without further analysis. An incomplete dfa is a variant of a dfa which is still very near to a complete dfa but has already gone a tiny step in direction of an nfa (as defined in Description 2.32 below).

Remark 2.20. Although the above theorem is published by Anil Nerode [67], it is general known as the Theorem of Myhill and Nerode and both scientists, John Myhill and Anil Nerode, are today acknowledged for this discovery. The notion of a derivative was fully investigated by Brzozowski when working on regular expressions [8].

Example 2.21. If $L = 0^*1^*2^*$ then $L_0 = 0^*1^*2^*$, $L_{01} = 1^*2^*$, $L_{012} = 2^*$ and $L_{0121} = \emptyset$. Every further L_x is equivalent to one of these four: If $x \in 0^*$ then $L_x = L$; if $x \in 0^*1^+$ then $L_x = 1^*2^*$ as a 0 following a 1 makes the word to be outside L; if $x \in 0^*1^*2^+$ then $L_x \in 2^*$. If $x \notin 0^*1^*2^*$ then also all extensions of x are outside L and $L_x = \emptyset$. The automaton obtained by the construction of Myhill and Nerode is the following.



As $L_{0121} = \emptyset$, one could also omit this node and would get an incomplete dfa with all states being accepting. Then a word is accepted as long as one can go on in the automaton on its symbols.

Example 2.22. Consider the language $\{0^n 1^n : n \in \mathbb{N}\}$. Then $L_{0^n} = \{0^m 1^{m+n} : m \in \mathbb{N}\}$ is unique for each $n \in \mathbb{N}$. Hence, if this language would be recognised by a dfa, then the dfa would need infinitely many states, what is impossible.

Lemma 2.23: Jaffe's Matching Pumping Lemma [45]. A language $L \subseteq \Sigma^*$ is regular iff there is a constant k such that for all $x \in \Sigma^*$ and $y \in \Sigma^k$ there are u, v, w with $y = uvw$ and $v \neq \varepsilon$ such that, for all $h \in \mathbb{N}$, $L_{xuv^h w} = L_{xy}$.

Jaffe's Pumping Lemma as a Game. Anke wants to show that a given language L satisfies the pumping condition of Jaffe's Lemma and Boris wants to prove the opposite.

1. Anke selects a pumping constant k;
2. Boris selects a word xy with $|y| = k$;
3. Anke splits y into u, v, w with $v \neq \varepsilon$ and $y = uvw$;
4. Boris selects $h \in \mathbb{N}$ and $z \in \Sigma^*$;
5. If $L(xyz) = L(xuv^h wz)$ then Anke wins the game else Boris wins the game.

Note that $L_{xy} = L_{xuv^h w}$ iff for all z, $L(xyz) = L(xuv^h wz)$. Thus Boris tries to select a word z which witnesses a difference in the derivatives and Anke wins if the witness does not verify this.

Comment. This version of Jaffe's Pumping Lemma pumps at the end of the word xy inside the y-part. There is also an equivalent version where one pumps within the first k digits.

Proof. Assume that L satisfies Jaffe's Matching Pumping Lemma with constant k. For every word z with $|z| \geq k$ there is a splitting of z into xy with $|y| = k$. Now there is a shorter word xuw with $L_{xuw} = L_{xy}$; thus one can find, by repeatingly using this argument, that every derivative L_z is equal to some derivative $L_{z'}$ with $|z'| < k$. Hence there are only $1 + |\Sigma| + \ldots + |\Sigma|^{k-1}$ many different derivatives and therefore the language is regular by the Theorem of Myhill and Nerode.

The converse direction follows by considering a dfa recognising L and letting k be larger than the number of states in the dfa. Then when the dfa processes a word xyz and $|y| = k$, then there is a splitting of y into uvw with $v \neq \varepsilon$ such that the dfa is in the same state when processing xu and xuv. It follows that the dfa is, for every h, in the same state when processing xuv^h and therefore it accepts xuv^hwz iff it accepts xyz. Thus $L_{xuv^hw} = L_{xy}$ for all h. ▮

Exercise 2.24. *Assume that the alphabet Σ has 5000 elements. Define a language $L \subseteq \Sigma^*$ such that Jaffe's Matching Pumping Lemma is satisfied with constant $k = 3$ while every deterministic finite automaton recognising L has more than 5000 states. Prove the answer.*

Exercise 2.25. *Find a language which needs for Jaffe's Matching Pumping Lemma at least constant $k = 100$ and can be recognised by a deterministic finite automaton with 100 states. Prove the answer.*

Consider the following weaker version of Jaffe's Pumping Lemma which follows from it.

Corollary 2.26. *Regular languages L and also some others satisfy the following condition:*

There is a constant k such that for all $x \in \Sigma^*$ and $y \in \Sigma^k$ with $xy \in L$ there are u, v, w with $y = uvw$ and $v \neq \varepsilon$ such that, for all $h \in \mathbb{N}$, $L_{xuv^hw} = L_{xy}$.

That is, in Corollary 2.26, one postulates the property of Jaffe's Pumping Lemma only for members of L. Then it loses its strength and is no longer matching.

Exercise 2.27. *Show that the language $L = \{\varepsilon\} \cup \{0^n1^m2^k3 : n = m \text{ or } k = 0\}$ is a context-free language which satisfies Corollary 2.26 but is not regular. Furthermore, show directly that this language does not satisfy Jaffe's Pumping Lemma itself; this is expected, as only regular languages satisfy it.*

Exercise 2.28. *Is the following statement true: If L satisfies Corollary 2.26 and H is regular then $L \cdot H$ satisfies Corollary 2.26?*

Exercise 2.29. *Call a language prefix-free if whenever $vw \in L$ and $w \neq \varepsilon$ then $v \notin L$. Does every prefix-free language L for which L^{mi} satisfies Theorem 1.19 (a) also satisfy Corollary 2.26? Here x^{mi} is the mirror image of x, so $01122^{mi} = 22110$ and $L^{mi} = \{x^{mi} : x \in L\}$. Prove the answer.*

Exercise 2.30. *Let $\Sigma = \{0, 1, 2\}$. Call a word v square-containing iff it has a non-empty subword of the form ww with $w \in \Sigma^+$ and let L be the language of all square-containing words; call a word v palindrome-containing iff it has a non-empty subword of the form ww^{mi} or waw^{mi} with $a \in \Sigma$ and $w \in \Sigma^+$ and let H be the language* *of all palindrome-containing words.*

Are the languages L and H regular? If so, provide a dfa. Which of the pumping lemmas (except for the block pumping lemma) do they satisfy?

There are quite simple tasks where an automaton to check this might become much larger than it is adequate for the case. For example, to check whether a string contains a symbol twice, one would guess which symbol is twice and then just verify that it occurs twice; however, a deterministic finite automaton cannot do it and the following example provides a precise justification. Therefore, this chapter will look into mechanisms to formalise this intuitive approach which is to look at a word like 0120547869 where one, by just looking at it, might intuitively see that the 0 is double and then verify it with a closer look. Such type of intuition is not possible to a deterministic finite automaton; however, nondeterminism permits to model intuitive decisions as long as their is a way to make sure that the intuitive insight is correct (like scanning the word for the twice occurring letter).

Example 2.31. Assume that Σ has n elements. Consider the set L of all strings which contain at least one symbol at least twice.

There are at least $2^n + 1$ sets of the form L_x: If $x \in L$ then $L_x = \Sigma^*$ else $\varepsilon \notin L_x$. Furthermore, for $x \notin L$, $\Sigma \cap L_x = \{a \in \Sigma : a \text{ occurs in } x\}$. As there are 2^n subsets of Σ, one directly gets that there are 2^n states of this type.

On the other hand, one can also see that $2^n + 1$ is an upper bound. If the dfa has not seen any symbol twice so far then it just has to remember which symbols it has seen else the automaton needs just one additional state to go when it has seen some symbol twice. Representing the first states by the corresponding subsets of Σ and the second state by the special symbol #, the dfa would has the following parameters: $Q = Pow(\Sigma) \cup \{\#\}$, Σ is the alphabet, $\emptyset$ is the starting state and # is the unique final state. Furthermore, δ is is given by three cases: if $A \subseteq \Sigma$ and $a \in \Sigma - A$ then $\delta(A, a) = A \cup \{a\}$, if $A \subseteq \Sigma$ and $a \in A$ then $\delta(A, a) = \#$, $\delta(\#, a) = \#$.

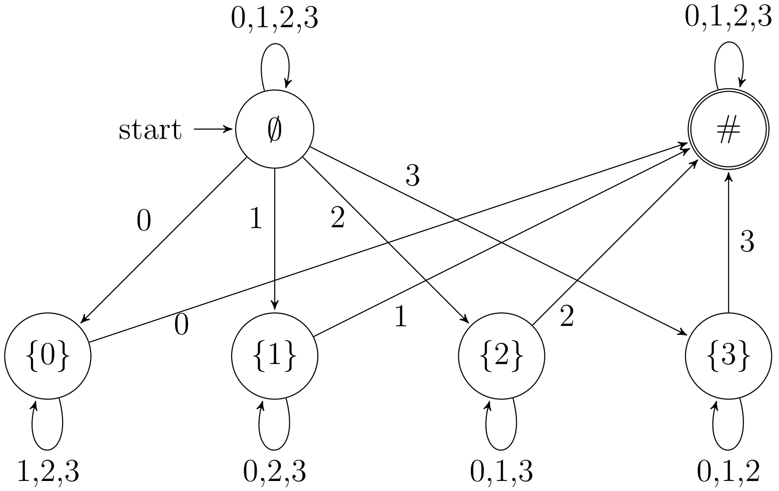
Description 2.32: Nondeterministic Finite Automaton. A nondeterministic automaton can guess information and, in the case that it guessed right, verify that a word is accepting.

A nondeterministic automaton $(Q, \Sigma, \delta, s, F)$ differs from the deterministic automaton in the way that δ is a multi-valued function, that is, for each $q \in Q$ and $a \in \Sigma$ the value $\delta(q, a)$ is a set of states.

Now one defines the acceptance-condition using the notion of a run: One says a string $q_0 q_1 \ldots q_n \in Q^{n+1}$ is a run of the automaton on input $a_1 \ldots a_n$ iff $q_0 = s$ and $q_{m+1} \in \delta(q_m, a_{m+1})$ for all $m \in \{1, \ldots, n\}$; note that the run has one symbol more than the string processed. The nondeterministic automaton accepts a word w iff there is a run on the input w whose last state is accepting.

Note that for accepting a word, there needs only to be at least one accepting run; other rejecting runs might also exist. For rejecting a word, all runs which exist must be rejecting, this includes the case that there is no run at all (neither an accepting nor a rejecting).

Example 2.33: Large DFA and small NFA. For the dfa with $2^n + 1$ states from Example 2.31, one can make an nfa with $n + 2$ states (here for $n = 4$ and $\Sigma = \{0, 1, 2, 3\}$). Thus an nfa can be exponentially smaller than a corresponding dfa.



In general, Q contains $\emptyset$ and $\{a\}$ for all $a \in \Sigma$ and $\#$; $\delta(\emptyset, a) = \{\emptyset, \{a\}\}$; $\delta(\{a\}, b)$ is $\{a\}$ in the case $a \neq b$ and is $\#$ in the case $a = b$; $\delta(\#, a) = \#$; $\emptyset$ is the starting state; $\#$ is the only accepting state.

So the nfa has $n+2$ and the dfa has 2^n+1 states (which cannot be made better). So the actual size of the dfa is more than a quarter of the theoretical upper bound 2^{n+2} which will be given by the construction found by Büchi [9, 10] as well as Rabin and Scott [72]. Their general construction which permits to show that every nfa with n states is equivalent to a dfa with 2^n states, that is, the nfa and the dfa constructed recognise the same language.

Theorem 2.34: Determinisation of NFAs [9, 10, 72]**.** *For each nfa $(Q, \Sigma, \delta, s, F)$ with $n = |Q|$ states, there is an equivalent dfa whose 2^n states are the subsets Q' of Q, whose starting state is $\{s\}$, whose update-function δ' is given by $\delta'(Q', a) = \{q'' \in Q : \exists q' \in Q' \, [q'' \in \delta(q', a)]\}$ and whose set of accepting states is $F' = \{Q' \subseteq Q : Q' \cap F \neq \emptyset\}$.*

Proof. It is clear that the automaton defined in the statement of the theorem is a dfa: For each set $Q' \subseteq Q$ and each $a \in \Sigma$, the function δ' selects a unique successor

$Q'' = \delta'(Q', a)$. Note that Q'' can be the empty set and that, by the definition of δ', $\delta'(\emptyset, a) = \emptyset$.

Assume now that the nfa accepts a word $w = a_1 a_2 \ldots a_m$ of m letters. Then there is an accepting run $(q_0, q_1, \ldots, q_m)$ on this word with $q_0 = s$ and $q_m \in F$. Let $Q_0 = \{s\}$ be the starting state of the dfa and, inductively, $Q_{k+1} = \delta'(Q_k, a_{k+1})$ for $k = 0, 1, \ldots, m-1$. One can verify by induction that $q_k \in Q_k$ for all $k \in \{0, 1, \ldots, m\}$: This is true for $q_0 = s$ by definition of Q_0; for the inductive step, if $q_k \in Q_k$ and $k < m$, then $q_{k+1} \in \delta(q_k, a_{k+1})$ and therefore $q_{k+1} \in Q_{k+1} = \delta'(Q_k, a_{k+1})$. Thus $Q_m \cap F$ contains the element q_m and therefore Q_m is an accepting state in the dfa.

For the converse direction on a given word $w = a_1 a_2 \ldots a_m$, assume that the run $(Q_0, Q_1, \ldots, Q_m)$ of the dfa on this word is accepting. Thus there is $q_m \in Q_m \cap F$. Now one can, inductively for $k = m-1, m-2, \ldots, 2, 1, 0$ choose a q_k such that $q_{k+1} \in \delta(q_k, a_{k+1})$ by the definition of δ'. It follows that $q_0 \in Q_0$ and therefore $q_0 = s$. Thus the so defined sequence $(q_0, q_1, \ldots, q_m)$ is an accepting run of the nfa on the word w and the nfa accepts the word w as well.

This shows that the dfa is equivalent to the nfa, that is, it accepts and it rejects the same words. Furthermore, as an n-element set has 2^n subsets, the dfa has 2^n states. ▮

Note that this construction produces, in many cases, too many states. Thus one would consider only those states (subsets of Q) which are reached from others previously constructed; in some cases this can save a lot of work. Furthermore, once the dfa is constructed, one can run the algorithm of Myhill and Nerode to make a minimal dfa out of the constructed one.

Example 2.35. Consider the nfa $(\{s, q\}, \{0, 1\}, \delta, s, \{q\})$ with $\delta(s, 0) = \{s, q\}$, $\delta(s, 1) = \{s\}$ and $\delta(q, a) = \emptyset$ for all $a \in \{0, 1\}$.

Then the corresponding dfa has the four states $\emptyset, \{s\}, \{q\}, \{s, q\}$ where $\{q\}, \{s, q\}$ are the final states and $\{s\}$ is the initial state. The transition function δ' of the dfa is given as

$\delta'(\emptyset, a) = \emptyset$ for $a \in \{0, 1\}$,
$\delta'(\{s\}, 0) = \{s, q\}$, $\delta'(\{s\}, 1) = \{s\}$,
$\delta'(\{q\}, a) = \emptyset$ for $a \in \{0, 1\}$,
$\delta'(\{s, q\}, 0) = \{s, q\}$, $\delta'(\{s, q\}, 1) = \{s\}$.

This automaton can be further optimised: The states $\emptyset$ and $\{q\}$ are never reached, hence they can be omitted from the dfa.

The next exercise shows that the exponential blow-up between the nfa and the dfa is also there when the alphabet is fixed to $\Sigma = \{0, 1\}$.

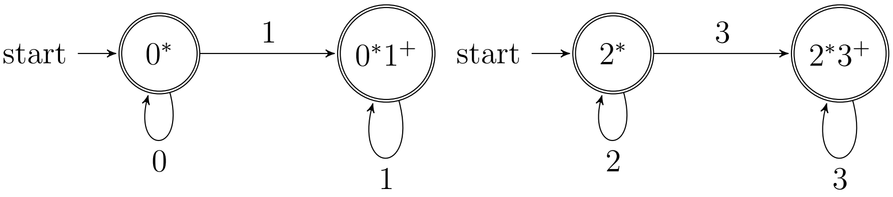
Exercise 2.36. *Consider the language* $\{0,1\}^* \cdot 0 \cdot \{0,1\}^{n-1}$*:*
(a) *Show that a dfa recognising it needs at least* 2^n *states;*
(b) *Make an nfa recognising it with at most* $n+1$ *states;*
(c) *Made a dfa recognising it with exactly* 2^n *states.*

Exercise 2.37. *Find a characterisation when a regular language* L *is recognised by an nfa only having accepting states. Examples of such languages are* $\{0,1\}^*$, $0^*1^*2^*$ *and* $\{1, 01, 001\}^* \cdot 0^*$*. The language* $\{00, 11\}^*$ *is not a language of this type.*

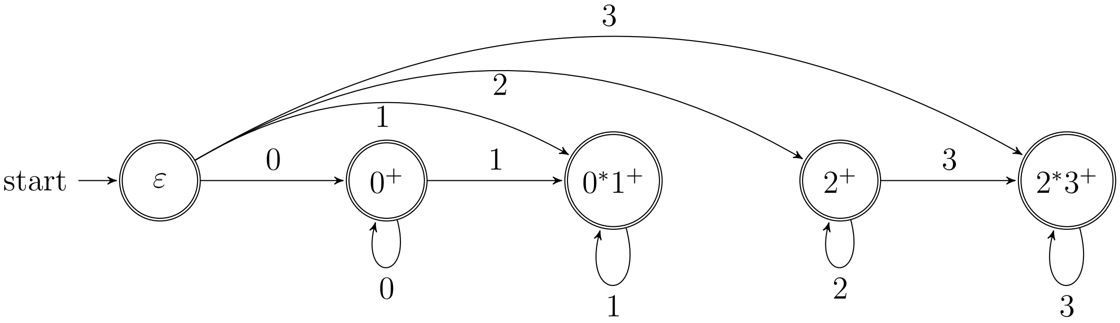
Example 2.38. One can generalise the nfa to a machine $(Q, \Sigma, \delta, I, F)$ where a set I of starting states replaces the single starting state s. Now such a machine accepts a string $w = a_1 a_2 \ldots a_i \in \Sigma^i$ iff there is a sequence $q_0 q_1 \ldots q_i$ of states such that

$$q_0 \in I \wedge q_i \in F \wedge \forall j < i \, [q_{i+1} \in \delta(q_i, a_i)];$$

if such a sequence does not exist then the machine rejects the input w. The following machine with three states recognises the set $0^*1^* \cup 2^*3^*$, the nodes are labelled with the regular expressions denoting the language of the words through which one can reach the corresponding node.



The corresponding nfa would need 5 states, as one needs a common start state which the nfa leaves as soon as it reads a symbol.



Exercise 2.39. *Let* $\Sigma = \{0, 1, \ldots, n-1\}$ *and* $L = \{w \in \Sigma^* :$ *some* $a \in \Sigma$ *does not occur in* $w\}$*. Show that there is a machine like in Example 2.38 with* $|Q| = n$ *which recognises* L *and that every complete dfa recognising* L *needs* 2^n *states.*

The exact trade-off between the numbers of states of an nfa and of a complete dfa was determined by Meyer and Fischer [65]. Their construction does not need the above multiple start states.

Exercise 2.40. *Given an nfa* $(\{q_0, q_1, \ldots, q_{n-1}\}, \{0,1\}, \delta, q_0, \{q_0\})$ *with* $\delta(q_m, 1) = \{q_{(m+1)\, mod\, n}\}$*,* $\delta(q_0, 0) = \emptyset$ *and* $\delta(q_m, 0) = \{q_0, q_m\}$ *for* $m \in \{1, 2, \ldots, n-1\}$*. Determine the number of states of an equivalent complete and minimal dfa and explain how this number is derived.*

Exercise 2.41. *Assume that the alphabet is unary, that is,* $\Sigma = \{0\}$*. Now show that every nfa with two states over this alphabet has an equivalent dfa with up to three states. For this, carry out the Büchi construction and show that at least one state is not reached.*

Theorem 2.42. *Every language generated by a regular grammar is also recognised by an nfa.*

Proof. If a grammar has a rule of the form $A \to w$ with w being non-empty, one can add a non-terminal B and replace the rule $A \to w$ by $A \to wB$ and $B \to \varepsilon$. Furthermore, if the grammar has a rule $A \to a_1 a_2 \ldots a_n B$ with $n > 1$ then one can introduce $n-1$ new non-terminals $C_1, C_2, \ldots, C_{n-1}$ and replace the rule by $A \to a_1 C_1$, $C_1 \to a_2 C_2$, …, $C_{n-1} \to a_n B$. Thus if L is regular, there is a grammar (N, Σ, P, S) generating L such that all rules in P are either of the form $A \to B$ or the form $A \to aB$ or of the form $A \to \varepsilon$ where $A, B \in N$ and $a \in \Sigma$. So let such a grammar be given.

Now an nfa recognising L is given as $(N, \Sigma, \delta, S, F)$ where N and S are as in the grammar and for $A \in N, a \in \Sigma$, one defines

$$\begin{aligned} \delta(A, a) &= \{B \in N : A \Rightarrow^* aB \text{ in the grammar}\}; \\ F &= \{B \in N : B \Rightarrow^* \varepsilon\}. \end{aligned}$$

If now $w = a_1 a_2 \ldots a_n$ is a word in L then there is a derivation of the word $a_1 a_2 \ldots a_n$ of the form

$$S \Rightarrow^* a_1 A_1 \Rightarrow^* a_1 a_2 A_2 \Rightarrow^* \ldots \Rightarrow^* a_1 a_2 \ldots a_{n-1}\, A_{n-1} \Rightarrow^* a_1 a_2 \ldots a_{n-1} a_n A_n \Rightarrow^* a_1 a_2 \ldots a_n.$$

In particular, $S \Rightarrow^* a_1 A_1$, $A_m \Rightarrow^* a_{m+1} A_{m+1}$ for all $m \in \{1, 2, \ldots, n-1\}$ and $A_n \Rightarrow^* \varepsilon$. It follows that A_n is an accepting state and $(S, A_1, A_2, \ldots, A_n)$ an accepting run of the nfa on the word $a_1 a_2 \ldots a_n$.

If now the nfa has an accepting run $(S, A_1, A_2, \ldots, A_n)$ on a word $w = a_1 a_2 \ldots a_n$

then $S \Rightarrow^* a_1 A_1$ and, for all $m \in \{1, 2, \ldots, n-1\}$, $A_m \Rightarrow^* a_{m+1} A_{m+1}$ and $A_n \Rightarrow^* \varepsilon$. It follows that $w \in L$ as witnessed by the derivation $S \Rightarrow^* a_1 A_1 \Rightarrow^* a_1 a_2 A_2 \Rightarrow^* \ldots \Rightarrow^* a_1 a_2 \ldots a_{n-1} A_{n-1} \Rightarrow^* a_1 a_2 \ldots a_{n-1} a_n A_n \Rightarrow^* a_1 a_2 \ldots a_n$. Thus the nfa constructed recognises the language L. ▮

Example 2.43. The language $L = 0123^*$ has a grammar with terminal alphabet $\Sigma = \{0, 1, 2, 3\}$, non-terminal alphabet $\{S, T\}$, start symbol S and rules $S \to 012|012T$, $T \to 3T|3$.

One first updates the grammar such that all rules are of the form $A \to aB$ or $A \to \varepsilon$ for $A, B \in N$ and $a \in \Sigma$. One possible updated grammar has the non-terminals $N = \{S, S', S'', S''', T, T'\}$, the start symbol S and the rules $S \to 0S'$, $S' \to 1S''$, $S'' \to 2S'''|2T$, $S''' \to \varepsilon$, $T \to 3T|3T'$, $T' \to \varepsilon$.

Now the nondeterministic finite automaton is given as $(N, \Sigma, \delta, S, \{S''', T\})$ where $\delta(S, 0) = \{S'\}$, $\delta(S', 1) = \{S''\}$, $\delta(S'', 2) = \{S''', T'\}$, $\delta(T, 3) = \{T, T'\}$ and $\delta(A, a) = \emptyset$ in all other cases.

Examples for accepting runs: For 0 1 2, an accepting run is $S\,(0)\,S'\,(1)\,S''\,(2)\,S'''$ and for 0 1 2 3 3 3, an accepting run is $S\,(0)\,S'\,(1)\,S''\,(2)\,T\,(3)\,T\,(3)\,T\,(3)\,T'$.

Exercise 2.44. *Let the regular grammar $(\{S, T\}, \{0, 1, 2\}, P, S)$ with the rules P being $S \to 01T|20S$, $T \to 01|20S|12T$. Construct a nondeterministic finite automaton recognising the language generated by this grammar.*

Exercise 2.45. *Consider the regular grammar $(\{S\}, \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}, P, S)$ where the rules in P are all rules of the form $S \to aaaaaS$ for some digit a and the rule $S \to \varepsilon$ and let L be the language generated by this grammar. What is the minimum number of states of a nondeterministic finite automaton recognising this language L? What is the trade-off of the nfa compared to the minimal dfa for the same language L? Prove the answers.*

Theorem 1.14 showed that a language L is generated by a regular expression iff it has a regular grammar; Theorem 2.8 showed that if L is recognised by a dfa then it L is also generated by a regular expression; Theorem 2.34 showed that if L is recognised by an nfa then L is recognised by a dfa; Theorem 2.42 showed if L is generated by a regular grammar then L is recognised by an nfa. Thus these four concepts are all equivalent.

Corollary 2.46. *A language L is regular iff it satisfies any of the following equivalent conditions:*

(a) *L is generated by a regular expression;*
(b) *L is generated by a regular grammar;*

(c) *L is recognised by a deterministic finite automaton;*
(d) *L is recognised by a nondeterministic finite automaton;*
(e) *L and its complement satisfy both the block pumping lemma;*
(f) *L satisfies Jaffe's pumping lemma;*
(g) *L has only finitely many derivatives (Theorem of Myhill and Nerode).*

It was shown above that deterministic automata can be exponentially larger than nondeterministic automata in the sense that a nondeterministic automaton with n states can only be translated into a deterministic complete automaton with 2^n states, provided that one permits multiple start states. One might therefore ask, how do the other notions relate to the size of states of automata. For the sizes of regular expressions, they depend heavily on the question of which operation one permits. Gelade and Neven [34] showed that not permitting intersection and complement in regular expressions can cause a double exponential increase in the size of the expression (measured in number of symbols to write down the expression).

Example 2.47. The language $L = \bigcup_{m<n}(\{0,1\}^m \cdot \{1\} \cdot \{0,1\}^* \cdot \{10^m\})$ can be written down in $O(n^2)$ symbols as a regular expression but the corresponding dfa has at least 2^n states: if $x = a_0a_1 \ldots a_{n-1}$ then $10^m \in L_x$ iff $x10^m \in L$ iff $a_0a_1 \ldots a_{n-1}10^m \in L$ iff $a_m = 1$. Thus for $x = a_0a_1 \ldots a_{n-1}$ and $y = b_0b_1 \ldots b_{n-1}$, it holds that $L_x = L_y$ iff $\forall m < n\,[10^m \in L_x \Leftrightarrow 10^m \in L_y]$ iff $\forall m < n\,[a_m = b_m]$ iff $x = y$. Thus the language L has at least 2^n derivatives and therefore a dfa for L needs at least 2^n states.

One can separate regular expressions with intersections even from nfas over the unary alphabet $\{0\}$ as the following theorem shows; for this theorem, let $p_1, p_2, \ldots, p_n$ be the first n prime numbers.

Theorem 2.48. *The language $L_n = \{0^{p_1}\}^+ \cap \{0^{p_2}\}^+ \cap \ldots \cap \{0^{p_n}\}^+$ has a regular expression which can be written down with approximately $O(n^2 \log(n))$ symbols if one can use intersection. However, every nfa recognising L_n has at least 2^n states and every regular expression for L_n only using union, concatenation and Kleene star needs at least 2^n symbols.*

Proof. It is known that $p_n \leq 2 \cdot n \cdot \log(n)$ for almost all n. Each set 0^{p_m} can be written down as a regular expression consisting of two set brackets and p_m zeroes in between, if one uses Kleene star and not Kleene plus, one uses about $2p_m + 6$ symbols (two times 0^{p_m} and four set brackets and one star and one concatenation symbol, where Kleene star and plus bind stronger than concatenation, union and intersection). The n terms are then put into brackets and connected with intersection symbols what gives a total of up to $2n \cdot p_n + 3n$ symbols. So the overall number of symbols is $O(n^2 \log(n))$ in dependence of the parameter n.

The shortest word in the language must be a word of the form 0^k where each of the prime numbers $p_1, p_2, \ldots, p_n$ divides k; as all of them are distinct primes, their product is at least 2^n and the product divides k, thus $k \geq 2^n$. In an nfa, the length of the shortest accepted word is as long as the shortest path to an accepting state; in this path, each state is visited at most once and therefore the length of the shortest word is smaller than the number of states. It follows that an nfa recognising L must have at least 2^n states.

If a regular expression generating at least one word and only consisting of listed finite sets connected with union, concatenation, Kleene plus and Kleene star, then one can prove that the shortest word generated by σ is at most as long as the length of the expression. By way of contradiction, assume that σ be the length-lexicographically first regular expression such that σ generates some words, but all of these are longer than σ. Let $sw(\sigma)$ denote the shortest word generated by σ (if it exists) and if there are several, $sw(\sigma)$ is the lexicographically first of those.

- If σ is a list of words of a finite set, no word listed can be longer than σ, thus $|sw(\sigma)| \leq |\sigma|$.
- If $\sigma = (\tau \cup \rho)$ then at least one of τ, ρ is non-empty, say τ. As $|\tau| < |\sigma|$, $|sw(\tau)| \leq |\tau|$. Now $|sw(\sigma)| \leq |sw(\tau)| \leq |\tau| \leq |\sigma|$.
- If $\sigma = (\tau \cdot \rho)$ then $|\tau|, |\rho| < |\sigma|$ and $|sw(\sigma)| = |sw(\tau)| + |sw(\rho)|$, as the shortest words generated by τ and ρ concatenated give the shortest word generated by σ. It follows that $|sw(\tau)| \leq |\tau|$, $|sw(\rho)| \leq |\rho|$ and $|sw(\sigma)| = |sw(\tau)| + |sw(\rho)| \leq |\tau| + |\rho| \leq |\sigma|$.
- If $\sigma = \tau^*$ then $\varepsilon = sw(\sigma)$ and clearly $|sw(\sigma)| \leq |\sigma|$.
- If $\sigma = \tau^+$ then $sw(\sigma) = sw(\tau)$ and $|\tau| < |\sigma|$, thus $|sw(\sigma)| = |sw(\tau)| \leq |\tau| \leq |\sigma|$.

Thus in all five cases the shortest word generated by σ is at most as long as σ. It follows that any regular expression generating L and consisting only of finite sets, union, concatenation, Kleene star and Kleene plus must be at least 2^n symbols long. ▮

Example 2.49: Inductive Definition of Shortest Word. A counterpart to structural induction are inductive definitions, which can also run along the structure of regular expressions. For this, recall that for an alphabet Σ, the length-lexicographic order chooses the shorter string, if two strings v, w compared are not of the same length, and the lexicographically first string in the case that both strings have the same length. So, for $\Sigma = \{0, 1\}$, the order is $\varepsilon <_{ll} 0 <_{ll} 1 <_{ll} 00 <_{ll} 01 <_{ll} 10 <_{ll} 11 <_{ll} 000 <_{ll} \ldots$ and one uses this length-lexicographical order $<_{ll}$ to define the shortest word of a regular experssion $sw(reg\,exp)$. The inductive definition over the structure of regular expressions follows the following case-distinction:

$$sw(\emptyset) \;=\; \infty;$$

$$
\begin{array}{rcl}
sw(\{w_1,\ldots,w_n\}) & = & min_{ll}\{w_1,\ldots,w_n\}; \\
sw(\sigma\cup\tau) & = & \begin{cases} sw(\sigma) & \text{if } sw(\tau)=\infty; \\ sw(\tau) & \text{if } sw(\sigma)=\infty; \\ min_{ll}\{sw(\sigma),sw(\tau)\} & \text{otherwise;} \end{cases} \\
sw(\sigma\cdot\tau) & = & \begin{cases} \infty & \text{if } sw(\sigma)=\infty \\ & \text{or } sw(\tau)=\infty; \\ sw(\sigma)\cdot sw(\tau) & \text{otherwise;} \end{cases} \\
sw(\sigma^*) & = & \varepsilon.
\end{array}
$$

Now one could also use this structural definition to prove along the above cases that $|sw(\sigma)| \leq |\sigma|$ where $\{,\},(,),\emptyset,\cup,\cdot,^*,\infty$ are extra symbols not in Σ which are used in either regular expressions or the output to denote that the regular expression does not produce a word. Furthermore, $|\varepsilon| = 0$. In listings of finite sets, one denotes the empty string by just making a string of length 0 over Σ, for example the input $\{,00,001\}$ to sw stands for $\{\varepsilon,00,001\}$. Note that $\{\}$ is therefore $\{\varepsilon\}$ and not $\emptyset$. It is left to the reader to adjust the above definition and the treatment of regular expressions such that brackets are taking correctly into account.

Exercise 2.50. *Assume that a regular expression uses lists of finite sets, Kleene star, union and concatenation and assume that this expression generates at least two words. Prove that the second-shortest word of the language generated by σ is at most as long as σ. Either prove it by structural induction or by an assumption of contradiction as in the proof before; both methods are nearly equivalent.*

Exercise 2.51. *Is Exercise 2.50 also true if one permits Kleene plus in addition to Kleene star in the regular expressions? Either provide a counter example or adjust the proof. In the case that it is not true for the bound $|\sigma|$, is it true for the bound $2|\sigma|$? Again prove that bound or provide a further counter example.*

Example 2.52: Ehrenfeucht and Zeiger's Exponential Gap [27]**.** Assume that the alphabet Σ consists of all pairs of numbers in $\{1,2,\ldots,n\}\times\{1,2,\ldots,n\}$. Then a complete dfa with $n+1$ states accepts all sequences of the form $(1,a_1),(a_1,a_2),(a_2,a_3),\ldots,(a_{m-1},a_m)$ for any numbers $a_1,a_2,\ldots,a_m$, where the automaton has the following transition-function: If it is in state a on input (a,b) then it goes to state b else it goes to state 0. The starting state is 1; the set $\{1,2,\ldots,n\}$ is the set of accepting states and once it reaches the state 0, the automaton never leaves this state. Ehrenfeucht and Zeiger showed that any regular expression for this language needs at least 2^{n-1} symbols.

If one would permit intersection, this gap would not be there for this example, as one could write

$(\{(a,b)\cdot(b,c) : a,b,c \in \{1,2,\ldots,n\}\}^* \cdot (\varepsilon \cup \{(a,b) : a,b \in \{1,2,\ldots,n\}\}))$
$\cap\ (\{(1,b) : b \in \{1,2,\ldots,n\}\} \cdot \{(a,b)\cdot(b,c) : a,b,c \in \{1,2,\ldots,n\}\}^* \cdot (\varepsilon \cup$
$\{(a,b) : a,b \in \{1,2,\ldots,n\}\}))$

to obtain the desired expression whose size is polynomial in n.

Exercise 2.53. *Assume that an nfa of k states recognises a language L. Show that the language does then satisfy the Block Pumping Lemma (Theorem 2.9) with constant $k+1$, that is, given any words $u_0, u_1, \ldots, u_k, u_{k+1}$ such that their concatenation $u_0 u_1 \ldots u_k u_{k+1}$ is in L then there are i, j with $0 < i < j \leq k+1$ and*

$$u_0 u_1 \ldots u_{i-1} (u_i u_{i+1} \ldots u_{j-1})^* u_j u_{j+1} \ldots u_{k+1} \subseteq L.$$

Exercise 2.54. *Given numbers n, m with $n > m > 2$, provide an example of a regular language where the Block pumping constant is exactly m and where every nfa needs at least n states.*

In the following five exercises, one should try to find small nfas; however, full marks are also awarded if the nfa is small but not the smallest possible.

Exercise 2.55. *Consider the language $H = \{vawa : v, w \in \Sigma^*, a \in \Sigma\}$. Let n be the size of the alphabet Σ and assume $n \geq 2$. Determine the size of the smallest dfa of H in dependence of n and give a good upper bound for the size of the nfa. Explain the results and construct the automata for $\Sigma = \{0, 1\}$.*

Exercise 2.56. *Consider the language $I = \{ua : u \in (\Sigma - \{a\})^*, a \in \Sigma\}$. Let n be the size of the alphabet Σ and assume $n \geq 2$. Determine the size of the smallest dfa of I in dependence of n and give a good upper bound for the size of the nfa. Explain the results and construct the automata for $\Sigma = \{0, 1\}$.*

Exercise 2.57. *Consider the language $J = \{abuc : a, b \in \Sigma, u \in \Sigma^*, c \in \{a, b\}\}$. Let n be the size of the alphabet Σ and assume $n \geq 2$. Determine the size of the smallest dfa of J in dependence of n and give a good upper bound for the size of the nfa. Explain the results and construct the automata for $\Sigma = \{0, 1\}$.*

Exercise 2.58. *Consider the language $K = \{avbwc : a, b \in \Sigma, v, w \in \Sigma^*, c \notin \{a, b\}\}$. Let n be the size of the alphabet Σ and assume $n \geq 2$. Determine the size of the smallest dfa of K in dependence of n and give a good upper bound for the size of the nfa. Explain the results and construct the automata for $\Sigma = \{0, 1\}$.*

Exercise 2.59. *Consider the language $L = \{w : \exists\, a, b \in \Sigma\, [w \in \{a, b\}^*]\}$. Let n be the size of the alphabet Σ and assume $n \geq 2$. Determine the size of the smallest dfa* *of L in dependence of n and give a good upper bound for the size of the nfa. Explain the results and construct the automata for $\Sigma = \{0, 1, 2\}$.*

The next exercises deal with Jaffe's Pumping Lemma and its constants.

Exercise 2.60. *Show that an nfa for the language $\{0000000\}^* \cup \{00000000\}^*$ needs only 16 states while the constant for Jaffe's pumping lemma is 56.*

Exercise 2.61. *Generalise the idea of Exercise 2.60 to show that there is a family L_n of languages such that an nfa for L_n can be constructed with $O(n^3)$ states while Jaffe's pumping lemma needs a constant of at least 2^n. Provide the family of the L_n and explain why it satisfies the corresponding bounds.*

Exercise 2.62. *Determine the constant of Jaffe's pumping lemma and the sizes of minimal nfa and dfa for $(\{00\} \cdot \{00000\}) \cup (\{00\}^* \cap \{000\}^*)$.*

3 Combining Languages

One can form new languages from old ones by combining them with basic set-theoretical operations. In most cases, the complexity in terms of the level of the Chomsky hierarchy does not change.

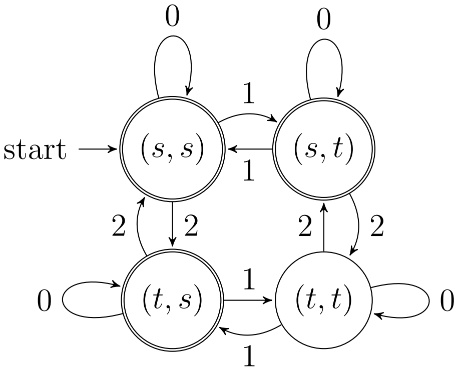
Theorem 3.1: Basic Closure Properties. *Assume that L, H are languages which are on the level CHk of the Chomsky hierarchy. Then the following languages are also on the level CHk: $L \cup H$, $L \cdot H$ and L^*.*

Description 3.2: Transforming Regular Expressions into Automata. First it is shown how to form dfas which recognise the intersection, union or difference of given sets. So let $(Q_1, \Sigma, \delta_1, s_1, F_1)$ and $(Q_2, \Sigma, \delta_2, s_2, F_2)$ be dfas which recognise L_1 and L_2, respectively.

Let $(Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, (s_1, s_2), F)$ with $(\delta_1 \times \delta_2)((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$ be a product automaton of the two given automata; here one can choose F such that it recognises the union or intersection or difference of the respective languages:

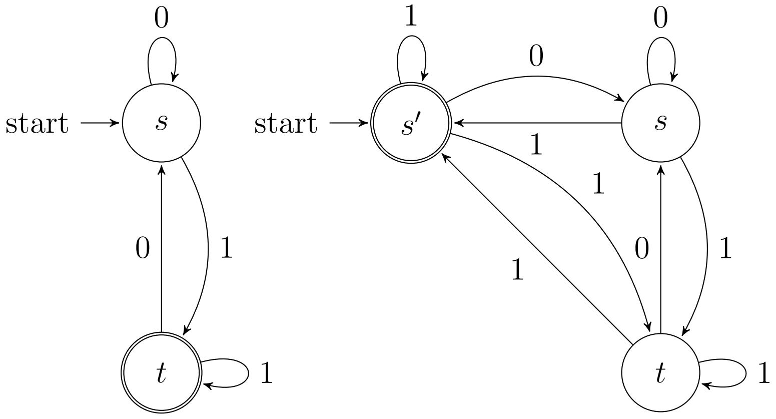
- Union: $F = F_1 \times Q_2 \cup Q_1 \times F_2$;
- Intersection: $F = F_1 \times F_2 = F_1 \times Q_2 \cap Q_1 \times F_2$;
- Difference: $F = F_1 \times (Q_2 - F_2)$;
- Symmetric Difference: $F = F_1 \times (Q_2 - F_2) \cup (Q_1 - F_1) \times F_2$.

For example, let the first automaton recognise the language of words in $\{0, 1, 2\}$ with an even number of 1s and the second automaton with an even number of 2s. Both automata have the accepting and starting state s and a rejection state t; they change between s and t whenever they see 1 or 2, respectively. The product automaton is now given as follows:



The automaton given here recognises the union. For the other operations like Kleene star and concatenation, one needs to form an nfa recognising the corresponding language first and can then use Büchi's construction to transform the nfa into a dfa; as every dfa is an nfa, one can directly start with an nfa.

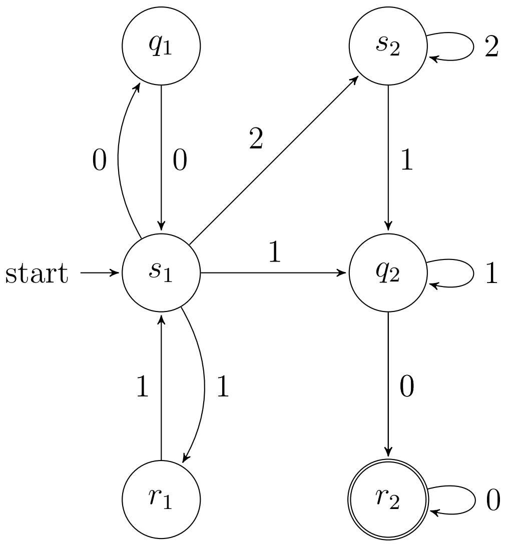
So assume $(Q, \Sigma, \delta, s, F)$ is an nfa recognising L. Now L^* is recognised by $(Q \cup \{s'\}, \Sigma, \delta', s', \{s'\})$ where $\delta' = \delta \cup \{(s', a, p) : (s, a, p) \in \delta\} \cup \{(p, a, s) : (p, a, q) \in \delta$ for some $q \in F\} \cup \{(s', a, s') : a \in L\}$. The last part of the union is to add all one-symbol words from L. This automaton has a new starting state s' which is accepting, as $\varepsilon \in L^*$. The other states in Q are kept so that the automaton can go through the states in Q in order to simulate the original automaton on some word w until it is going to process the last symbol when it then returns to s'; so it can process sequences of words in Q each time going through s'. After the last word w_n of $w_1 w_2 \ldots w_n \in L^*$, the automaton can either return to s' in order to accept the word. Here an example.



The next operation with nfas is the Concatenation. Here assume that $(Q_1, \Sigma, \delta_1, s_1, F_1)$ and $(Q_2, \Sigma, \delta_2, s_2, F_2)$ are nfas recognising L_1 and L_2 with $Q_1 \cap Q_2 = \emptyset$ and assume $\varepsilon \notin L_2$. Now $(Q_1 \cup Q_2, \Sigma, \delta, s_1, F_2)$ recognises $L_1 \cdot L_2$ where $(p, a, q) \in \delta$ whenever $(p, a, q) \in \delta_1 \cup \delta_2$ or $p \in F_1 \wedge (s_2, a, q) \in \delta_2$.

Note that if L_2 contains ε then one can consider the union of L_1 and $L_1 \cdot (L_2 - \{\varepsilon\})$.

An example is the following: $L_1 \cdot L_2$ with $L_1 = \{00, 11\}^*$ and $L_2 = 2^*1^+0^+$.



Last but not least, one has to see how to build an automaton recognising a finite set, as the above only deal with the question how to get a new automaton recognising unions, differences, intersections, concatenations and Kleene star of given regular languages represented by their automata. For finite sets, one can simply consider all possible derivatives (which are easy to compute from a list of strings in the language) and then connect the corresponding states accordingly. This would indeed give the smallest dfa recognising the corresponding set.

Alternatively, one can make an automaton recognising the set $\{w\}$ and then form product automata for the unions in order to recognise sets of several strings. Here a dfa recognising $\{a_1 a_2 \ldots a_n\}$ for such a string of n symbols would have the states $q_0, q_1, \ldots, q_n$ plus r and go from q_m to q_{m+1} on input a_{m+1} and in all other cases would go to state r. Only the state q_n is accepting.

Exercise 3.3. *The above gives upper bounds on the size of the dfa for a union, intersection, difference and symmetric difference as n^2 states, provided that the original two dfas have at most n states. Give the corresponding bounds for nfas: If L and H are recognised by nfas having at most n states each, how many states does one need at most for an nfa recognising (a) the union $L \cup H$, (b) the intersection $L \cap H$, (c) the difference $L - H$ and (d) the symmetric difference $(L - H) \cup (H - L)$? Give the bounds in terms of "linear", "quadratic" and "exponential". Explain the bounds.*

Exercise 3.4. *Let $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Construct a (not necessarily complete) dfa recognising the language $(\Sigma \cdot \{aa : a \in \Sigma\}^*) \cap \{aaaaa : a \in \Sigma\}^*$. It is not needed to give a full table for the dfa, but a general schema and an explanation how it works.*

Exercise 3.5. *Make an nfa for the intersection of the following languages:* $\{0,1,2\}^* \cdot \{001\} \cdot \{0,1,2\}^* \cdot \{001\} \cdot \{0,1,2\}^*$*;* $\{001,0001,2\}^*$*;* $\{0,1,2\}^* \cdot \{00120001\} \cdot \{0,1,2\}^*$*.*

Exercise 3.6. *Make an nfa for the union* $L_0 \cup L_1 \cup L_2$ *with* $L_a = \{0,1,2\}^* \cdot \{aa\} \cdot \{0,1,2\}^* \cdot \{aa\} \cdot \{0,1,2\}^*$ *for* $a \in \{0,1,2\}$*.*

Exercise 3.7. *Consider two context-free grammars with terminals* Σ*, disjoint non-terminals* N_1 *and* N_2*, start symbols* $S_1 \in N_1$ *and* $S_2 \in N_2$ *and rule sets* P_1 *and* P_2 *which generate* L *and* H*, respectively. Explain how to form from these a new context-free grammar for* **(a)** $L \cup H$*,* **(b)** $L \cdot H$ *and* **(c)** L^**.*

Write down the context-free grammars for $\{0^n 1^{2n} : n \in \mathbb{N}\}$ *and* $\{0^n 1^{3n} : n \in \mathbb{N}\}$ *and form the grammars for the union, concatenation and star explicitly.*

Example 3.8. The language $L = \{0^n 1^n 2^n : n \in \mathbb{N}\}$ is the intersection of the context-free languages $\{0\}^* \cdot \{1^n 2^n : n \in \mathbb{N}\}$ and $\{0^n 1^n : n \in \mathbb{N}\} \cdot \{2\}^*$. By Exercise 1.27 this language is not context-free.

Hence L is the intersection of two context-free languages which is not context-free. However, the complement of L is context-free. The following grammar generates $\{0^k 1^m 2^n : k < n\}$: the non-terminals are S, T with S being the start symbol, the terminals are $0, 1, 2$ and the rules are $S \to 0S2|S2|T2$, $T \to 1T|\varepsilon$. Now the complement of L is the union of eight context-free languages. Six languages of this type: $\{0^k 1^m 2^n : k < m\}$, $\{0^k 1^m 2^n : k > m\}$, $\{0^k 1^m 2^n : k < n\}$, $\{0^k 1^m 2^n : k > n\}$, $\{0^k 1^m 2^n : m < n\}$ and $\{0^k 1^m 2^n : m > n\}$; furthermore, the two regular languages $\{0,1,2\}^* \cdot \{10, 20, 21\} \cdot \{0,1,2\}^*$ and $\{\varepsilon\}$. So the so-constructed language is context-free while its complement L itself is not.

Although the intersection of two context-free languages might not be context-free, one can still show a weaker version of this result. This weaker version can be useful for various proofs.

Theorem 3.9. *Assume that* L *is a context-free language and* H *is a regular language. Then the intersection* $L \cap H$ *is also a context-free language.*

Proof. Assume that (N, Σ, P, S) is the context-free grammar generating L and $(Q, \Sigma, \delta, s, F)$ is the finite automaton accepting H. Furthermore, assume that every production in P is either of the form $A \to BC$ or of the form $A \to w$ for $A, B, C \in N$ and $w \in \Sigma^*$.

Now make a new grammar $(Q \times N \times Q \cup \{S\}, \Sigma, R, S)$ generating $L \cap H$ with the following rules:

- $S \to (s, S, q)$ for all $q \in F$;

- $(p, A, q) \to (p, B, r)(r, C, q)$ for all $p, q, r \in Q$ and all rules of the form $A \to BC$ in P;
- $(p, A, q) \to w$ for all $p, q \in Q$ and all rules $A \to w$ in P with $\delta(p, w) = q$.

For each $A \in N$, let $L_A = \{w \in \Sigma^* : A \Rightarrow^* w\}$. For each $p, q \in Q$, let $H_{p,q} = \{w \in \Sigma^* : \delta(p, w) = q\}$. Now one shows that (p, A, q) generates w in the new grammar iff $w \in L_A \cap H_{p,q}$.

First one shows by induction over every derivation-length that a symbol (p, A, q) can only generate a word w iff $\delta(p, w) = q$ and $w \in L_A$. If the derivation-length is 1 then there is a production $(p, A, q) \to w$ in the grammar. It follows from the definition that $\delta(p, w) = q$ and $A \to w$ is a rule in P, thus $w \in L_A$. If the derivation-length is larger than 1, then one uses the induction hypothesis that the statement is already shown for all shorter derivations and now looks at the first rule applied in the derivation. It is of the form $(p, A, q) \to (q, B, r)(r, C, q)$ for some $B, C \in N$ and $r \in Q$. Furthermore, there is a splitting of w into uv such that (q, B, r) generates u and (r, C, q) generates v. By induction hypothesis and the construction of the grammar, $u \in L_B$, $v \in L_C$, $\delta(p, u) = r$, $\delta(r, v) = q$ and $A \to BC$ is a rule in P. It follows that $A \Rightarrow BC \Rightarrow^* uv$ in the grammar for L and $w \in L_A$. Furthermore, $\delta(p, uv) = \delta(r, v) = q$, hence $w \in H_{p,q}$. This completes the proof of this part.

Second one shows that the converse holds, now by induction over the length of derivations in the grammar for L. Assume that $w \in L_A$ and $w \in H_{p,q}$. If the derivation has length 1 then $A \to w$ is a rule the grammar for L. As $\delta(p, w) = q$, it follows that $(p, A, q) \to w$ is a rule in the new grammar. If the derivation has length $n > 1$ and the proof has already been done for all derivations shorter than n, then the first rule applied to show that $w \in L_A$ must be a rule of the form $A \to BC$. There are $u \in L_B$ and $v \in L_C$ with $w = uv$. Let $r = \delta(p, u)$. It follows from the definition of δ that $q = \delta(r, v)$. Hence, by induction hypothesis, (p, B, r) generates u and (r, C, q) generates v. Furthermore, the rule $(p, A, q) \to (p, B, r)(r, C, q)$ is in the new grammar, hence (p, A, q) generates $w = uv$.

Now one has for each $p, q \in Q$, $A \in N$ and $w \in \Sigma^*$ that (p, A, q) generates w iff $w \in L_A \cap H_{p,q}$. Furthermore, in the new grammar, S generates a string w iff there is a $q \in F$ with (s, S, q) generating w iff $w \in L_S$ and $\delta(s, w) \in F$ iff $w \in L_S$ and there is a $q \in F$ with $w \in H_{s,q}$ iff $w \in L \cap H$. This completes the proof. ▮

Exercise 3.10. *Recall that the language L of all words which contain as many 0s as 1s is context-free; a grammar for it is $(\{S\}, \{0, 1\}, \{S \to SS|\varepsilon|0S1|1S0\}, S)$. Construct a context-free grammar for $L \cap (001^+)^*$.*

Exercise 3.11. *Let again L be the language of all words which contain as many 0s as 1s. Construct a context-free grammar for $L \cap 0^*1^*0^*1^*$.*

Theorem 3.12. *The concatenation of two context-sensitive languages is context-sensitive.*

Proof. Let L_1 and L_2 be context-sensitive languages not containing ε and consider context-sensitive grammars (N_1, Σ, P_1, S_1) and (N_2, Σ, P_2, S_2) generating L_1 and L_2, respectively, where $N_1 \cap N_2 = \emptyset$ and where each rule $l \to r$ satisfies $|l| \leq |r|$ and $l \in N_e^+$ for the respective $e \in \{1, 2\}$. Let $S \notin N_1 \cup N_2 \cup \Sigma$. Now the automaton

$$(N_1 \cup N_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S)$$

generates $L_1 \cdot L_2$: If $v \in L_1$ and $w \in L_2$ then $S \Rightarrow S_1 S_2 \Rightarrow^* v S_2 \Rightarrow^* vw$. Furthermore, the first rule has to be $S \Rightarrow S_1 S_2$ and from then onwards, each rule has on the left side either $l \in N_1^*$ so that it applies to the part generated from S_1 or it has in the left side $l \in N_2^*$ so that l is in the part of the word generated from S_2. Hence every intermediate word z in the derivation is of the form $xy = z$ with $S_1 \Rightarrow^* x$ and $S_2 \Rightarrow^* y$.

In the case that one wants to form $(L_1 \cup \{\varepsilon\}) \cdot L_2$, one has to add the rule $S \to S_2$, for $L_1 \cdot (L_2 \cup \{\varepsilon\})$, one has to add the rule $S \to S_1$ and for $(L_1 \cup \{\varepsilon\}) \cdot (L_2 \cup \{\varepsilon\})$, one has to add the rules $S \to S_1 | S_2 | \varepsilon$ to the grammar. ▮

As an example consider the following context-sensitive grammars generating two sets L_1 and L_2 not containing the empty string ε, the second grammar could also be replaced by a context-free grammar but is here only chosen to be context-sensitive:

- $(\{S_1, T_1, U_1, V_1\}, \{0, 1, 2, 3, 4\}, P_1, S_1)$ with P_1 containing the rules $S_1 \to T_1 U_1 V_1 S_1 \mid T_1 U_1 V_1$, $T_1 U_1 \to U_1 T_1$, $T_1 V_1 \to V_1 T_1$, $U_1 T_1 \to T_1 U_1$, $U_1 V_1 \to V_1 U_1$, $V_1 T_1 \to T_1 V_1$, $V_1 U_1 \to U_1 V_1$, $T_1 \to 0$, $V_1 \to 1$, $U_1 \to 2$ generating all words with the same nonzero number of 0s, 1s and 2s;
- $(\{S_2, T_2, U_2\}, \{0, 1, 2, 3, 4\}, P_2, S_2)$ with P_2 containing the rules $S_2 \to U_2 T_2 S_2 \mid U_2 T_2$, $U_2 T_2 \to T_2 U_2$, $T_2 U_2 \to U_2 T_2$, $U_2 \to 3$, $T_2 \to 4$ generating all words with the same nonzero number of 3s and 4s.

The grammar $(\{S, S_1, T_1, U_1, V_1, S_2, T_2, U_2\}, \{0, 1, 2, 3, 4\}, P, S)$ with P containing $S \to S_1 S_2$, $S_1 \to T_1 U_1 V_1 S_1 | T_1 U_1 V_1$, $T_1 U_1 \to U_1 T_1$, $T_1 V_1 \to V_1 T_1$, $U_1 T_1 \to T_1 U_1$, $U_1 V_1 \to V_1 U_1$, $V_1 T_1 \to T_1 V_1$, $V_1 U_1 \to U_1 V_1$, $T_1 \to 0$, $V_1 \to 1$, $U_1 \to 2$, $S_2 \to U_2 T_2 S_2 | U_2 T_2$, $U_2 T_2 \to T_2 U_2$, $T_2 U_2 \to U_2 T_2$, $U_2 \to 3$, $T_2 \to 4$ generates all words with consisting of n 0s, 1s and 2s in any order followed by m 3s and 4s in any order with $n, m > 0$. For example, 01120234434334 is a word in this language. The grammar is context-sensitive in the sense that $|l| \leq |r|$ for all rules $l \to r$ in P.

Theorem 3.13. *If L is context-sensitive so is L^*.*

Proof. Assume that (N_1, Σ, P_1, S_1) and (N_2, Σ, P_2, S_2) are two context-sensitive grammars for L with $N_1 \cap N_2 = \emptyset$ and all rules $l \to r$ satisfying $|l| \leq |r|$ and $l \in N_1^+$ or

$l \in N_2^+$, respectively. Let S, S' be symbols not in $N_1 \cup N_2 \cup \Sigma$.

The new grammar is of the form $(N_1 \cup N_2 \cup \{S, S'\}, \Sigma, P, S)$ where P contains the rules $S \to S'|\varepsilon$ and $S' \to S_1S_2S' \mid S_1S_2 \mid S_1$ plus all rules in $P_1 \cup P_2$.

The overall idea is the following: if $w_1, w_2, \ldots, w_{2n}$ are non-empty words in L, then one generates $w_1w_2 \ldots w_{2n}$ by first generating the string $(S_1S_2)^n$ using the rule $S \to S'$, $n-1$ times the rule $S' \to S_1S_2S'$ and one time the rule $S' \to S_1S_2$. Afterwords one derives inductively S_1 to w_1, then the next S_2 to w_2, then the next S_1 to w_3, $\ldots$, until one has achieved that all S_1 and S_2 are transformed into the corresponding w_m.

The alternations between S_1 and S_2 are there to prevent that one can non-terminals generated for a word w_k and for the next word w_{k+1} mix in order to derive something what should not be derived. So only words in L^* can be derived. ▮

Exercise 3.14. *Recall that the language $L = \{0^n1^n2^n : n \in \mathbb{N}\}$ is context-sensitive. Construct a context-sensitive grammar for L^*.*

Theorem 3.15. *The intersection of two context-sensitive languages is context-sensitive.*

Proof Sketch. Let (N_k, Σ, P_k, S) be grammars for L_1 and L_2. Now make a new non-terminal set $N = (N_1 \cup \Sigma \cup \{\#\}) \times (N_2 \cup \Sigma \cup \{\#\})$ with start symbol $\binom{S}{S}$ and following types of rules:
(a) Rules to generate and manage space;
(b) Rules to generate a word v in the upper row;
(c) Rules to generate a word w in the lower row;
(d) Rules to convert a string from N into v provided that the upper components and lower components of the string are both v.

(a): $\binom{S}{S} \to \binom{S}{S}\binom{\#}{\#}$ for producing space; $\binom{A}{B}\binom{\#}{C} \to \binom{\#}{B}\binom{A}{C}$ and $\binom{A}{C}\binom{B}{\#} \to \binom{A}{\#}\binom{B}{C}$ for space management.

(b) and (c): For each rule in P_1, for example, for $AB \to CDE \in P_1$, and all symbols $F, G, H, \ldots$ in N_2, one has the corresponding rule $\binom{A}{F}\binom{B}{G}\binom{\#}{H} \to \binom{C}{F}\binom{D}{G}\binom{E}{H}$. So rules in P_1 are simulated in the upper half and rules in P_2 are simulated in the lower half and they use up # if the left side is shorter than the right one.

(d): Each rule $\binom{a}{a} \to a$ for $a \in \Sigma$ is there to convert a matching pair $\binom{a}{a}$ from $\Sigma \times \Sigma$ (a nonterminal) to a (a terminal).

The idea of the derivation of a word w is then to first use rules of type (a) to produce a string of the form $\binom{S}{S}\binom{\#}{\#}^{|w|-1}$ and afterwards to use the rules of type (b) to derive the word w in the upper row and the rules of type (c) to derive the word w in the lower row; these rules are used in combination with rules for moving # to the front in the upper or lower half. If both derivations have produced terminal words in the upper and lower half (terminals in the original grammar, not with respect to the new intersection grammar) and if these words match, then one can use the rules of type (d) which are $\binom{a}{a} \rightarrow a$ for terminals a to indeed derive w. However, if the derivations of the words in the upper row and lower row do not match, then the rules of type (d) cannot derive any terminal word, as there are symbols of the type $\binom{a}{b}$ for different terminals a, b in the original grammar. Thus only words in the intersection can be derived this way. If ε is in the derivation, some special rule can be added to derive ε directly from a new start state which can only be mapped to either ε or $\binom{S}{S}$ by a derivation rule. ▮

Example 3.16. *Let $Eq_{a,b}$ be the language of all non-empty words w over Σ such that w contains as many a as b where $a, b \in \Sigma$. Let $\Sigma = \{0, 1, 2\}$ and $L = Eq_{0,1} \cap Eq_{0,2}$. The language L is context-sensitive.*

Proof. First one makes a grammar for $Eq_{a,b}$ where c stands for any symbol in $\Sigma - \{a, b\}$. The grammar has the form

$$(\{S\}, \Sigma, \{S \rightarrow SS|aSb|bSa|ab|ba|c\}, S)$$

and one now makes a new grammar for the intersection as follows: The idea is to produce two-componented characters where the upper component belongs to a derivation of $Eq_{0,1}$ and the lower belongs to a derivation of $Eq_{0,2}$. Furthermore, there will in both components be a space symbol, #, which can be produced on the right side of the start symbol in the beginning and later be moved from the right to the left. Rules which apply only to the upper or lower component do not change the length, they just eat up some spaces if needed. Then the derivation is done on the upper and lower part independently. In the case that the outcome is on the upper and the lower component the same, the whole word is then transformed into the corresponding symbols from Σ.

The non-terminals of the new grammar are all of the form $\binom{A}{B}$ where $A, B \in \{S, \#, 0, 1, 2\}$. In general, each non-terminal represents a pair of a symbols which can occur in the upper and lower derivation; pairs are by definition different from terminals in $\Sigma = \{0, 1, 2\}$. The start symbol is $\binom{S}{S}$. The following rules are there:

1. The rule $\binom{S}{S} \rightarrow \binom{S}{S}\binom{\#}{\#}$. This rule permits to produce space right of the start symbol which is later used independently in the upper or lower component.
   For each symbols A, B, C in $\{S, \#, 0, 1, 2\}$ one introduces the rules $\binom{A}{B}\binom{\#}{C} \rightarrow \binom{\#}{B}\binom{A}{C}$ and $\binom{A}{C}\binom{B}{\#} \rightarrow \binom{A}{\#}\binom{B}{C}$ which enable to bring, independently of each other, the spaces in the upper and lower component from the right to the left.

2. The rules of $Eq_{0,1}$ will be implemented in the upper component. If a rule of the form $l \to r$ has that $|l| + k = |r|$ then one replaces it by $l\#^k \to r$. Furthermore, the rules have now to reflect the lower component as well, so there are entries which remain unchanged but have to be mentioned. Therefore one adds for each choice of $A, B, C \in \{S, \#, 0, 1, 2\}$ the following rules into the set of rules of the grammar:
   $\binom{S}{A}\binom{\#}{B} \to \binom{S}{A}\binom{S}{B}$, $\binom{S}{A}\binom{\#}{B}\binom{\#}{C} \to \binom{0}{A}\binom{S}{B}\binom{1}{C} \mid \binom{1}{A}\binom{S}{B}\binom{0}{C}$,
   $\binom{S}{A}\binom{\#}{B} \to \binom{0}{A}\binom{1}{B} \mid \binom{1}{A}\binom{0}{B}$, $\binom{S}{A} \to \binom{2}{A}$;
3. The rules of $Eq_{0,2}$ are implemented in the lower component and one takes again for all $A, B, C \in \{S, \#, 0, 1, 2\}$ the following rules into the grammar:
   $\binom{A}{S}\binom{B}{\#} \to \binom{A}{S}\binom{B}{S}$, $\binom{A}{S}\binom{B}{\#}\binom{C}{\#} \to \binom{A}{0}\binom{B}{S}\binom{C}{2} \mid \binom{A}{2}\binom{B}{S}\binom{C}{0}$,
   $\binom{A}{S}\binom{B}{\#} \to \binom{A}{0}\binom{B}{2} \mid \binom{A}{2}\binom{B}{0}$, $\binom{A}{S} \to \binom{A}{1}$;
4. To finalise, one has the rule $\binom{a}{a} \to a$ for each $a \in \Sigma$, that is, the rules $\binom{0}{0} \to 0$, $\binom{1}{1} \to 1$, $\binom{2}{2} \to 2$ in order to transform non-terminals consisting of matching placeholders into the corresponding terminals. Nonmatching placeholders and spaces cannot be finalised, if they remain in the word, the derivation cannot terminate.

To sum up, a word $w \in \Sigma^*$ can only be derived iff w is derived independently in the upper and the lower component of the string of non-terminals according to the rules of $Eq_{0,1}$ and $Eq_{0,2}$. The resulting string of pairs of matching entries from Σ is then transformed into the word w.

The following derivation of the word 011022 illustrates the way the word is generated: in the first step, enough space is produced; in the second step, the upper component is derived; in the third step, the lower component is derived; in the fourth step, the terminals are generated from the placeholders.

1. $\binom{S}{S} \Rightarrow \binom{S}{S}\binom{\#}{\#} \Rightarrow \binom{S}{S}\binom{\#}{\#}\binom{\#}{\#} \Rightarrow \binom{S}{S}\binom{\#}{\#}\binom{\#}{\#}\binom{\#}{\#} \Rightarrow \binom{S}{S}\binom{\#}{\#}\binom{\#}{\#}\binom{\#}{\#}\binom{\#}{\#} \Rightarrow$
   $\binom{S}{S}\binom{\#}{\#}\binom{\#}{\#}\binom{\#}{\#}\binom{\#}{\#}\binom{\#}{\#} \Rightarrow$
2. $\binom{S}{S}\binom{S}{\#}\binom{\#}{\#}\binom{\#}{\#}\binom{\#}{\#}\binom{\#}{\#} \Rightarrow \binom{S}{S}\binom{2}{\#}\binom{2}{\#}\binom{\#}{\#}\binom{\#}{\#}\binom{\#}{\#} \Rightarrow^* \binom{S}{S}\binom{\#}{\#}\binom{\#}{\#}\binom{\#}{\#}\binom{2}{\#}\binom{2}{\#} \Rightarrow$
   $\binom{S}{S}\binom{S}{\#}\binom{\#}{\#}\binom{\#}{\#}\binom{2}{\#}\binom{2}{\#} \Rightarrow \binom{S}{S}\binom{\#}{\#}\binom{S}{\#}\binom{\#}{\#}\binom{2}{\#}\binom{2}{\#} \Rightarrow \binom{0}{S}\binom{1}{\#}\binom{S}{\#}\binom{\#}{\#}\binom{2}{\#}\binom{2}{\#} \Rightarrow$
   $\binom{0}{S}\binom{1}{\#}\binom{1}{\#}\binom{0}{\#}\binom{2}{\#}\binom{2}{\#} \Rightarrow$
3. $\binom{0}{0}\binom{1}{S}\binom{1}{2}\binom{0}{\#}\binom{2}{\#}\binom{2}{\#} \Rightarrow^* \binom{0}{0}\binom{1}{S}\binom{1}{\#}\binom{0}{\#}\binom{2}{\#}\binom{2}{2} \Rightarrow \binom{0}{0}\binom{1}{S}\binom{1}{S}\binom{0}{\#}\binom{2}{\#}\binom{2}{2} \Rightarrow$
   $\binom{0}{0}\binom{1}{1}\binom{1}{S}\binom{0}{\#}\binom{2}{\#}\binom{2}{2} \Rightarrow \binom{0}{0}\binom{1}{1}\binom{1}{S}\binom{0}{S}\binom{2}{\#}\binom{2}{2} \Rightarrow \binom{0}{0}\binom{1}{1}\binom{1}{1}\binom{0}{S}\binom{2}{\#}\binom{2}{2} \Rightarrow$
   $\binom{0}{0}\binom{1}{1}\binom{1}{1}\binom{0}{0}\binom{2}{2}\binom{2}{2} \Rightarrow$

4. $0\binom{1}{1}\binom{1}{1}\binom{0}{0}\binom{2}{2}\binom{2}{2} \Rightarrow 01\binom{1}{1}\binom{0}{0}\binom{2}{2}\binom{2}{2} \Rightarrow 011\binom{0}{0}\binom{2}{2}\binom{2}{2} \Rightarrow 0110\binom{2}{2}\binom{2}{2} \Rightarrow$ $01102\binom{2}{2} \Rightarrow 011022$.

In this derivation, each step is shown except that several moves of characters in components over spaces are put together to one move. ▮

Exercise 3.17. *Consider the language* $L = \{00\} \cdot \{0,1,2,3\}^* \cup \{1,2,3\} \cdot \{0,1,2,3\}^* \cup \{0,1,2,3\}^* \cdot \{02,03,13,10,20,30,21,31,32\} \cdot \{0,1,2,3\}^* \cup \{\varepsilon\} \cup \{01^n2^n3^n : n \in \mathbb{N}\}$. *Which of the pumping conditions from Theorems 1.19 (a) and 1.19 (b), Corollary 1.20 and Theorem 2.9 does the language satisfy? Determine its exact position in the Chomsky hierarchy.*

Exercise 3.18. *Let* x^{mi} *be the mirror image of* x*, so* $(01001)^{mi} = 10010$*. Furthermore, let* $L^{mi} = \{x^{mi} : x \in L\}$*. Show the following two statements:*
(a) *If an nfa with* n *states recognises* L *then there is also an nfa with up to* $n+1$ *states recognising* L^{mi}*.*
(b) *Find the smallest nfas which recognise* $L = 0^*(1^* \cup 2^*)$ *as well as* L^{mi}*.*

Description 3.19: Palindromes. The members of the language $\{x \in \Sigma^* : x = x^{mi}\}$ are called palindromes. A palindrome is a word or phrase which looks the same from both directions.

An example is the German name “OTTO”; furthermore, when ignoring spaces and punctuation marks, a famous palindrome is the phrase “A man, a plan, a canal: Panama.” This palindrome was from Leigh Mercer (1893-1977), a British hobby-writer, who created lots of palindromes, Eckler [23] lists at the end of his article 100 of them.

The grammar with the rules $S \to aSa|aa|a|\varepsilon$ with a ranging over all members of Σ generates all palindromes; so for $\Sigma = \{0,1,2\}$ the rules of the grammar would be $S \to 0S0 \,|\, 1S1 \,|\, 2S2 \,|\, 00 \,|\, 11 \,|\, 22 \,|\, 0 \,|\, 1 \,|\, 2 \,|\, \varepsilon$.

The set of palindromes is not regular. This can easily be seen by the pumping lemma, as otherwise $L \cap 0^*10^* = \{0^n10^n : n \in \mathbb{N}\}$ would have to be regular. However, this is not the case, as there is a constant k such that one can pump the word 0^k10^k by omitting some of the first k characters; the resulting word 0^h10^k with $h < k$ is not in L as it is not a palindrome. Hence L does not satisfy the pumping lemma when the word has to be pumped among the first k characters.

Exercise 3.20. *Let* $w \in \{0,1,2,3,4,5,6,7,8,9\}^*$ *be a palindrome of even length and* n *be its decimal value. Prove that* n *is a multiple of* 11*. Note that it is essential that the length is even, as for odd length there are counter examples (like* 111 *and* 202*).*

Exercise 3.21. *Given a context-free grammar for a language L, is there also one for $L \cap L^{mi}$? If so, explain how to construct the grammar; if not, provide a counter example where L is context-free but $L \cap L^{mi}$ is not.*

Exercise 3.22. *Is the following statement true or false? Prove the answer: Given a language L, the language $L \cap L^{mi}$ equals to $\{w \in L : w$ is a palindrome$\}$.*

Exercise 3.23. *Let $L = \{w \in \{0,1,2\}^* : w = w^{mi}\}$ and consider $H = L \cap \{012, 210, 00, 11, 22\}^* \cap (\{0,1\}^* \cdot \{1,2\}^* \cdot \{0,1\}^*)$. This is the intersection of a context-free and regular language and thus context-free. Construct a context-free grammar for H.*

Definition 3.24. *Let $PUMP_{sw}$, $PUMP_{st}$ and $PUMP_{bl}$ be the classes of languages which can be pumped somewhere as formalised in Corollary 1.20, pumped at the start as formalised in Theorem 1.19 (a) and pumped according to a splitting in blocks as described in the Block Pumping Lemma (Theorem 2.9), respectively.*

The proof of Theorem 1.19 (a) showed that when L and H are in PUMP_{st} then so are $L \cup H$, $L \cdot H$, L^* and L^+.

Proposition 3.25. *The classes $PUMP_{sw}$ and $PUMP_{st}$ are closed under union, concatenation, Kleene star and Kleene plus.*

The next example establishes that these two classes are not closed under intersection, even not under intersection with regular languages.

Example 3.26. Consider $L = \{0^h 1^k 2^m 3^n : h = 0 \text{ or } k = m = n\}$ and consider $H = \{00\} \cdot \{1\}^* \cdot \{2\}^* \cdot \{3\}^*$. The language L is in PUMP_{sw} and PUMP_{st} and so is the language H as the latter is regular; however, the intersection $L \cap H = \{0^2 1^n 2^n 3^n : n \geq 0\}$ does not satisfy any pumping lemma. Furthermore, L^{mi} is not in PUMP_{st}.

Proposition 3.27. *If L is in $PUMP_{sw}$ so is L^{mi}; if L is in $PUMP_{bl}$ so is L^{mi}.*

Exercise 3.28. *Show that $PUMP_{bl}$ is closed under union and concatenation. Furthermore, show that the language $L = \{v3w4 : v, w \in \{0,1,2\}^*$ and if v, w are both square-free then $|v| \neq |w|$ or $v = w\}$ is in $PUMP_{bl}$ while L^+ and L^* are not.*

Theorem 3.29: Chak, Freivalds, Stephan and Tan [14]. *If L, H are in $PUMP_{bl}$ so is $L \cap H$.*

Proof. The proof uses Ramsey's Theorem of Pairs. Recall that when one splits a word x into blocks $u_0, u_1, \ldots, u_p$ then the borders between the blocks are called breakpoints;

furthermore, $u_1, \ldots, u_{p-1}$ should not be empty (otherwise one could pump the empty block).

Ramsey's Theorem of pairs says now that for every number c there is a number $c' > c$ such that given a word x with a set I of c' breakpoints, if one colours each pair (i, j) of breakpoints (pairs have always i strictly before j) in one of the colours "white" and "red", then one can select a subset $J \subseteq I$ of c breakpoints and a colour $q \in \{\text{white,red}\}$ such that each pair of the breakpoints in J has the colour q.

Now the idea is the following: Let c be a common upper bound of the two block pumping constants for L and H, this c is then also a valid block pumping constant. Then choose c' according to Ramsey's Theorem of Pairs and consider a word $x \in L \cap H$ split into $c' + 1$ parts by a set I of c' breakpoints. Now for each pair of breakpoints $i, j \in I$ splitting x into u, v, w, let the colour "white" denote that $u \cdot v^* \cdot w \subseteq L$ and "red" that this is not the case. By Ramsey's Theorem of Pairs there is a subset $J \subseteq I$ of c breakpoints which split x and a colour q such that each pair of breakpoints in J has colour q. As J consists of c breakpoints, there must be a pair (i, j) of breakpoints in J splitting x into $u \cdot v \cdot w$ with $u \cdot v^* \cdot w \subseteq L$, thus the colour q is white and therefore every pair of breakpoints in J has this property.

Now, as c is also the block pumping constant for H, there is a pair (i, j) of breakpoints in J which splits the word into u, v, w such that $u \cdot v^* \cdot w \subseteq H$. As seen before, $u \cdot v^* \cdot w \subseteq L$ and thus $u \cdot v^* \cdot w \subseteq L \cap H$. Thus $L \cap H$ is satisfies the Block Pumping Lemma with constant c' and $L \cap H$ is in PUMP$_{bl}$. ∎

A linear grammar is a grammar where every rule is either of the form $A \to u$ or of the form $A \to vBw$; here A, B are nonterminals and u, v, w are terminal words. Languages generated by linear grammars are called linear languages. An example for a linear language is the language of all palindromes; this example shows also that such languages do not need to be regular.

Exercise 3.30. *Show that the intersection of a linear language and a regular language is linear.*

A linear grammar is called balanced iff for every rule of the form $A \to vBw$ it holds that $|v| = |w|$ and a language is called balanced linear iff it is generated by a balanced linear grammar.

Exercise 3.31. *Is the intersection of two balanced linear languages again balanced linear? Prove the answer.*

Exercise 3.32. *Provide an example of a language which is linear but not balanced linear. Prove the answer.*

In the following, one considers regular expressions consisting of the symbol L of the language of palindromes over $\{0,1,2\}$ and the mentioned operations. What is the most difficult level in the hierarchy "regular, linear, context-free, context-sensitive" such expressions can generate. It can be used that the language $\{10^i10^j10^k1 : i \neq j, i \neq k, j \neq k\}$ is not context-free.

Exercise 3.33. *Determine the maximum possible complexity of the languages given by expressions containing L and $\cup$ and finite sets.*

Exercise 3.34. *Determine the maximum possible complexity of the languages given by expressions containing L and $\cup$ and $\cdot$ and Kleene star and finite sets.*

Exercise 3.35. *Determine the maximum possible complexity of the languages given by expressions containing L and $\cup$ and $\cdot$ and $\cap$ and Kleene star and finite sets.*

Exercise 3.36. *Determine the maximum possible complexity of the languages given by expressions containing L and $\cdot$ and set difference and Kleene star and finite sets.*

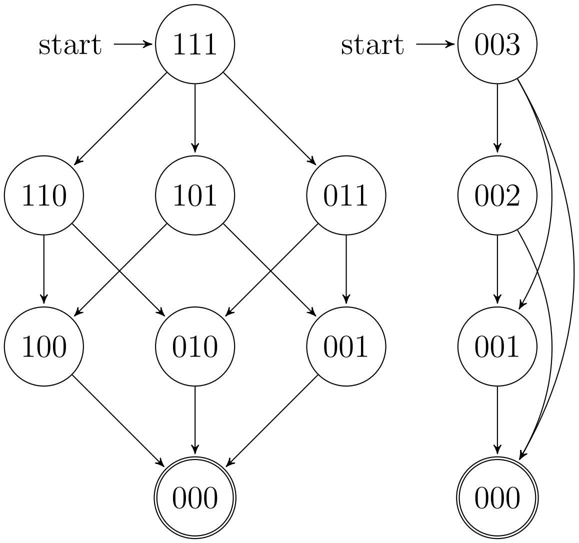
4 Games on Finite Graphs

The easiest game in a finite graph is a race to reach a member of a set of targets. Two players, Anke and Boris, move a marker alternately and that player who moves the marker first into a target-node wins the game, the game is supposed to start outside the set of targets. Without loss of generality, Anke is the player who moves first. So a game is given by a graph with a special set of vertices being the targets plus a starting-position of the marker (unless that is random). Furthermore, one might say that if a player ends up being unable to move, this player also loses the game.

Example 4.1. Consider a graph whose vertices are all labeled with 3-digit figures, so with $000, 001, \ldots, 999$, the start point is random. Now a player can move from ijk to $i'j'k'$ iff two digits are the same and the third digit is smaller than the previous one; the player who moves into 000 is the one who wins the game. The players move alternately.

Assume that 257 is the randomly chosen starting configuration. Now Anke moves $257 \to 157$. Boris replies by $157 \to 156$. Anke now moves $156 \to 116$. Boris replies $116 \to 110$. Now Anke moves $110 \to 100$ and Boris wins moving $100 \to 000$.

Assume now that 111 is the randomly chosen starting configuration. Then Anke wins: in each move, the number of 1s goes down by 1 and so Anke, Boris and then Anke can move where Anke reaches 000. For example Anke: $111 \to 110$; Boris: $110 \to 010$; Anke: $010 \to 000$. The game has a quite large graph, here just the small parts of the last play and the next one.



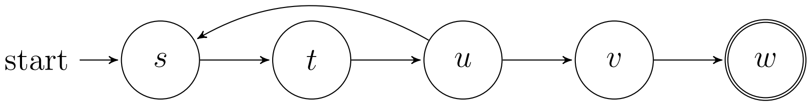
In the second play, starting at 003, Anke could win by $003 \to 000$; if she plays $003 \to 002$, Boris could win or make a bad move as well. So it depends on the move which player wins.

Definition 4.2: Winning Positions and Winning Strategies. *A winning strategy is an algorithm or table which tells Anke in each position how to move (in dependence of the prior moves which occurred in the game) such that Anke will eventually win. A node v is a winning position for Anke iff there is a winning strategy which tells Anke how to win, provided that the game starts from the node v. Similarly one defines winning strategies and positions for Boris.*

Example 4.3. In the game from Example 4.1, the node 111 is a winning position for each of the player (when it is his or her turn). The node 012 is also a winning position, as the player (whose turn it is) moves to 011; the opponent can only either move to 010 or 001 after which the player wins by moving to 000.

Exercise 4.4. *Consider the game from Example 4.1 and the following starting positions:* 123, 232, 330, 333. *Which of these starting positions are winning positions for Anke and which of these are winning positions for Boris? Explain the answer.*

Example 4.5. Consider the following game:



Each player who is in u cannot go directly to w but only to v; if the player decides, however, to go to v then the opponent would reach w and win the game. Therefore, if the player does not want to lose and is in u, the player would have to move to s. Thus the nodes s, t, u are not a winning position for either player, instead in these positions the player can preserve a draw. Such a draw might result in a play which runs forever; many board games have special rules to terminate the game as draw in the case that a situation repeats two or three times.

Several games (like the above) do not have that the starting position is a winning position for either player. Such games are called draw games.

Theorem 4.6. *There is an algorithm which determines which player has a winning strategy. The algorithm runs in time polynomial in the size of the graph.*

Proof. Let Q be the set of all nodes and T be the set of target nodes. The games starts in some node in $Q - T$.

1. Let $T_0 = T$ and $S_0 = \emptyset$ and $n = 0$.

2. Let $S_{n+1} = S_n \cup \{q \in Q - (T_n \cup S_n) :$ one can go in one step from q to a node in $T_n\}$.
3. Let $T_{n+1} = T_n \cup \{q \in Q - (T_n \cup S_{n+1}) :$ if one goes from q one step then one ends up in $S_{n+1}\}$.
4. If $S_{n+1} \neq S_n$ or $T_{n+1} \neq T_n$ then let $n = n + 1$ and goto 2.
5. Now S_n is the set of all winning positions for Anke and $T_n - T$ is the set of all winning positions for Boris and the remining nodes in $Q - (T_n \cup S_n)$ are draw positions.

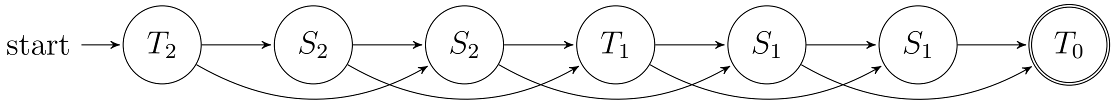
One can see that the algorithm terminates, as it can run the loop from steps 2 to 4 only as long as it adds nodes to the sets S_n or T_n, hence it runs at most $|Q|$ times through the loop.

Now one shows by induction that every set S_n consists of winning positions for Anke and T_n of winning positions for Boris. Clearly the nodes in S_1 permit Anke to win in one move. If Anke has to move from a node in $T_1 - T_0$ then she can either only move to nodes in S_1 or cannot move at all; in the first case, Boris wins the game (by symmetry), in the second case Anke loses the game as she cannot move.

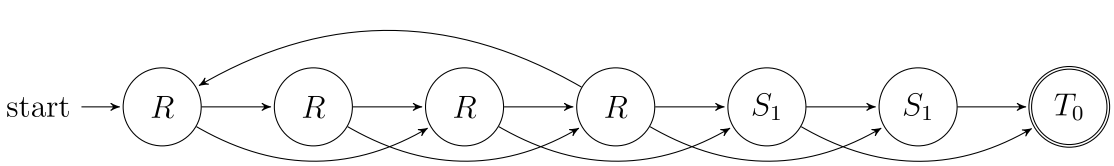
Assume now that the S_n consists of winning-positions for Anke and T_n of losing-positions for her, that is, winning-positions for Boris. Now S_{n+1} is the set of all nodes on which Anke can go into a node in T_n, that is, either Anke would win the game directly or Boris would lose it when continuing to play from that position. Hence every node in S_{n+1} is a winning-position for Anke. Furthermore, every node in T_{n+1} is a losing-position for Anke, for the nodes in T_n this is true by induction hypothesis and for the nodes in $T_{n+1} - T_n$, Anke can only move into a node in S_{n+1} from which on then Boris would win the game.

Hence, by induction, the final sets S_n are all winning positions for Anke and T_n are all winning-positions for Boris. Consider now any position in $q \in R$ with $R = Q - S_n - T_n$. Each node in R has at least one successor in R and every successor of it is either in R or in S_n. Hence the player (whether Anke or Boris) would move to a successor node in R and avoid going to S_n so that the opponent cannot win the game; as a result, the marker would circle indefinitely between the nodes in R.

The proof is illustrated by the following two examples of graphs. The nodes are labelled with the names of the sets to which they belong, therefore several nodes can have the same label, as they belong to the same set.

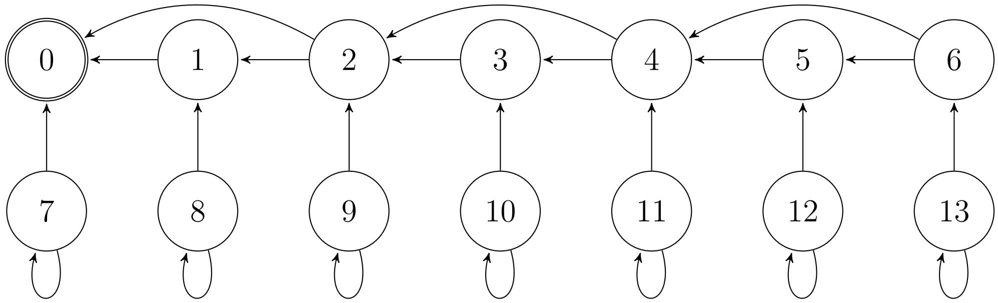


So the above game is a losing game for Anke and a winning game for Boris.



Here the players will always move inside the set R of nodes and not move to the nodes of S_1 as then the opponent wins. ▮

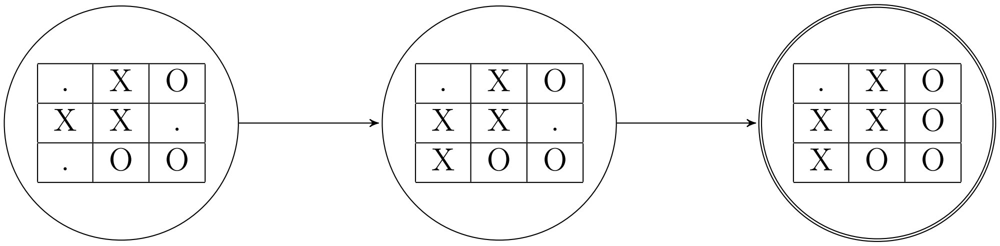
Exercise 4.7. *Consider a graph with node-set $Q = \{0, 1, 2, \ldots, 13\}$, target $T = \{0\}$ and the following edges between the nodes.*



Determine which of the nodes $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13$ are winning-positions, losing-positions and draw-positions for player Anke.

Example 4.8: Tic Tac Toe. Tic Tac Toe is a board game with a $3 * 3$ board. Anke and Boris place alternately an X and an O on the board until either there are three times the same symbol in a row, diagonal or column or all fields are full. In the case that a player makes the three symbols in a row / diagonal / column full, this player wins. Otherwise the game is a draw.

One can represent this as a game on a graph. Each possible board configuration represents a node and one makes an edge from one node to another one if the board of the second node is obtained from the board of the first node by placing one X or O into one field, respectively; furthermore, if there are as many X as O, an X has to be placed. If there are more X than O, an O has to be placed.



.	X	O
X	X	.
.	O	O

.	X	O
X	X	.
X	O	O

.	X	O
X	X	O
X	O	O

There are two more conditions to be taken care off: The starting configuration is the empty board and there is no outgoing edge in the case that the target has already been reached, that is, three symbols of the same type are already in a row / diagonal / column.

It should be noted that there are two types of nodes: Those nodes which equally many X and O are the nodes where Anke moves and places an X, those nodes with more X than O are the nodes where Boris moves and places an O.

One might ask how many nodes one needs to represent the game. An upper bound is certainly $3^9 = 19683$ which is the number of all $3 * 3$ boards with a space, X or O on each cell. So the graph of the game can easily be analysed by a computer. Furthermore, the number of nodes is even smaller as there are many cases which do not occur in a play, for example a board where there are all X in the top row and all O in the bottom row or a board with 5 X and 2 O in total. Indeed, the graph of the game has been analysed well and it has been shown that the game is a draw game; good human players can also always obtain a draw.

Description 4.9: Board Games. Many games have a default starting position and are played by two players with alternating moves where the set of possible configurations is finite, that is, such games can be represented as a graph game as done for Tic Tac Toe above. Traditionally, for board games with pieces, the starting player Anke has white pieces in the board game and the second player Boris the black pieces, so they are sometimes also referred as "White" or "First Player" and "Black" or "Second player", respectively. There are now three possible outcomes: The first player always wins the game when playing optimally, the second player always win the game when playing optimally, the game always ends up in a draw when both players play optimally. For some famous games it has been computed which player can force a win or whether the game ends up in a draw, see `http://en.wikipedia.org/wiki/Solved_game` for a current list of solved games and descriptions of the corresponding games. Here just an overview with the most famous games:

- The first player wins: Connect Four, Hex (on $n * n$ board), $15 * 15$ Gomoku (no opening rules).
- The second player wins: $4 * 4$ Othello, $6 * 6$ Othello.
- Draw: Draughts (also known as Checkers), Nine Men's Morris, Tic Tac Toe.
- Unknown: Chess, Go, $19 * 19$ Gomoku (conjecture: second player wins), $8 * 8$ Othello (conjecture: draw).

Gomoku is played on both sizes, $15 * 15$ and $19 * 19$; the latter is popular due to being the size of a Go board. The Wikipedia page on Gomoku has an example with a $15 * 15$

board. Opening rules are introduced in order to balance the chances for the winner and computers were not able to solve the balanced version of the $15 * 15$ Gomoku so far.

Furthermore, $8*8$ is the usual board size for Othello. Also this game has only been solved for smaller board sizes, so one does not know how the game would behave on the traditional size. Although an algorithm is known in principle, it uses up too much resources (computation time and space) to run on current computers. Nevertheless, computers can for some still unsolved games like chess compute strategies which are good although not optimal; such computer programs can defeat any human player, even the world chess champion Garry Kasparov was defeated by a computer in a tournament of six games in 1997.

Games involving random aspects (cards, dices, …) do not have perfect strategies. The reason is that a move which is good with high probability might turn out to be bad if some unlikely random event happens. Nevertheless, computers might be better than humans in playing these games.

Multiplayer games with more than two players usually do not have winning strategies as at three players, two of them might collaborate to avoid that the third player wins (although they should not do it). So it might be impossible for a single player to force a win.

Therefore the above analysis was for games with two players games without random aspects. If there is just a random starting point (in the graph), but no other random event, one can determine for each possible starting point which player has a winning strategy when starting from there.

Exercise 4.10. *Let $Divandinc_{n,m}$ be given by the graph with domain $\{1, 2, \ldots, n\}$, starting state $m \in \{2, \ldots, n\}$ and target state 1. Furthermore, each player can move from $k \in \{2, \ldots, n\}$ to $\ell \in \{1, 2, \ldots, n\}$ iff either $\ell = k+1$ or $\ell = k/p$ for some prime number p. Hence the game is called "$Divandinc_{n,m}$" (Divide and increment).*

(a) Show that there is no draw play, that is, whenever the game goes through an infinite sequence of moves then some player leaves out a possibility to win.

(b) Show that if $m \leq n \leq n'$ and n is a prime number, then the player who can win $Divandinc_{n,m}$ can also win $Divandinc_{n',m}$.

(c) Find values m, n, n' with $m < n < n'$ where Anke has a winning strategy for $Divandinc_{n,m}$ and Boris for $Divandinc_{n',m}$.

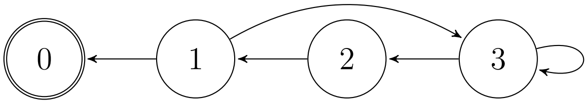
Remark 4.11. One can consider a variant of the game on finite graphs satisfying the following constraints:

- The set of nodes is partitioned into sets A, B such that every node is in exactly one of these sets and player Anke moves iff the marker is in A and player Boris moves iff the marker is in B;

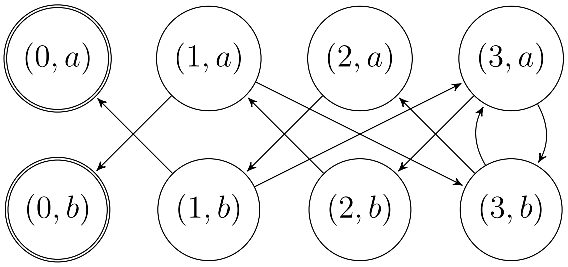
- There are three different disjoint sets T_A, T_B, T_D of target nodes and Anke wins whenever the game reaches one of the nodes in T_A and Boris wins whenever the game reaches one of the nodes in T_B and the game is draw when it ends up in T_D. Furthermore, a node is in $T_A \cup T_B \cup T_D$ iff it has no outgoing edges.

Tic Tac Toe, as described above, satisfies both constraints. A $3 * 3$ board is in A iff there are as many Xs as Os; a $3 * 3$-board is in B iff there are one X more than Os. Furthermore, T_A contains those boards from $A \cup B$ for which there is at least one row / diagonal / column with three Xs and none with three Os. T_B contains those boards from $A \cup B$ for which there is at least one row / diagonal / column with three Os and none with three Xs. T_D contains those $3*3$-boards where every field is either O or X but where there is no row, column or diagonal with all three symbols being the same. Boards with rows for both / diagonals / columns of three own symbols for both players cannot be reached in a play starting at the empty board without going through $T_A \cup T_B$, thus one can ignore those.

Example 4.12. Assume that a game with states Q and target set T is given. Now one can consider a new game with nodes $Q \times \{a, b\}$, there are the edges $(p, a) \to (q, b)$ and $(p, b) \to (q, a)$ in the new game whenever the edge $p \to q$ exists in the old game, $T_A = T \times \{b\}$, $T_B = T \times \{a\}$. Now $q_0 \to q_1 \to q_2 \to \ldots \to q_{2n} \to q_{2n+1}$ is a play in the old game iff $(q_0, a) \to (q_1, b) \to (q_2, a) \to \ldots \to (q_{2n}, a) \to (q_{2n+1}, b)$ is a play in the new game. Furthermore, that play is a winning play for Anke in the old game, that is, $q_{2n+1} \in T$ and no node before is in T, iff it is a winning play for Anke in the new game, that is, $(q_{2n+1}, a) \in T_A$ and no node before is in $T_A \cup T_B$. One can now show the following: A node $q \in T - Q$ is a winning position for Anke in the old game iff (q, a) is a winning position for Anke in the new game.



The above graph with $T = \{0\}$ is translated into the below one with $T_A = \{(0, b)\}$, $T_B = \{(0, a)\}$.



As $1, 3$ are the winning positions and 2 is a losing position in the old game, $(1, a), (3, a)$ are now winning positions and $(2, a)$ is a losing position for Anke in the new game. Furthermore, Boris wins from $(1, b), (3, b)$ and loses from $(2, b)$.

Exercise 4.13. *Design a game with A, B being disjoint nodes of Anke and Boris and the edges chosen such that*

- *the players move alternately;*
- *the sets T_A, T_B of the winning nodes are disjoint;*
- *every node outside $T_A \cup T_B$ has outgoing edges, that is, $T_D = \emptyset$;*
- *the so designed game is not an image of a symmetric game in the way it was done in the previous example.*

Which properties of the game can be used to enforce that?

Exercise 4.14. *The following game satisfies the second constraint from Remark 4.11 and has an infinite game graph.*

Assume that $Q = \mathbb{N}$, $x + 4, x + 3, x + 2, x + 1 \to x$ for all $x \in \mathbb{N}$ with the exception that nodes in T_A and T_B have no outgoing edges where $T_A = \{0, 6, 9\}$ and $T_B = \{5, 7, 12, 17\}$.

If the play of the game reaches a node in T_A then Anke wins and if it reaches a node in T_B then Boris wins. Note that if the game starts in nodes from T_A or T_B then it is a win for Anke or Boris in 0 moves, respectively.

Determine for both players (Anke and Boris) which are the winning positions for them. Are there any draw positions?

Alternation. Assume that an nfa is given and two players Anke and Boris. They process an input word w by alternatingly doing a move. Anke wins if the nfa is after the moves in an accepting state, Boris wins if the nfa is after the moves in a rejecting state.

This alternation game has been investigated in automata theory. However, it turned out that it is for many applications better if the two players do not move alternatingly but if there is an indication depending on state and character who moves. This gives rise to the notion of an alternating automaton.

Definition 4.15. *An alternating finite automaton (afa) is a finite automaton where for each pair q, a there are the following three possibilities:*

- *On (q, a) there is exactly one successor state q'. Then the automaton goes to q'.*
- *On (q, a) there is a disjunctive list like $q' \vee q'' \vee q'''$ of successor states. Then player Anke picks the successor among q', q'', q'''.*

- *On (q, a) there is a conjunctive list like $q' \wedge q'' \wedge q'''$ of successor states. Then player Boris picks the successor among q', q'', q'''.*

Anke wins a play of the automaton on a word w iff the automaton after all characters being processed is in an accepting state. An alternating automaton accepts a word w iff Anke has a winning strategy for w, that is, if Anke can win the game independent of whatever moves Boris makes when it is his turn to choose.

Example 4.16. Consider the alternating finite automaton with states $\{p, q, r\}$, alphabet $\{0, 1\}$ and the transition-rules as given by the following table:

state	type	0	1
p	start, rejecting	$p \wedge q \wedge r$	$q \vee r$
q	accepting	$p \wedge q \wedge r$	$p \vee r$
r	accepting	$p \wedge q \wedge r$	$p \vee q$

This alternating finite automaton accepts all words which ends with 1. To see this, consider any word w ending with 1. When the last 1 comes up, it is Anke to move. If the current state is p, she can either move to q or r, both are accepting; if the current state is q, she moves to the accepting state r; if the current state is r, she moves to the accepting state q. The empty word is rejected and a word with 0 is rejected as well. When the last digit 0 comes up, Boris moves and he can in all three cases choose the rejecting state p.

One can simulate an afa by an nfa. The idea is similar to Büchi's construction. Given an afa with a state Q of states, the states of the new nfa are the subsets of Q. When the nfa is in a state P, then the new state P' can be chosen by Anke as follows: For every $p \in P$, if the transition on (p, a) is to a conjunctive list $q_1 \wedge q_2 \wedge \ldots \wedge q_m$ then Anke has to put all the states $q_1, q_2, \ldots, q_m$ into P'; if the transition for (p, q) is a disjunctive list $q_1 \vee q_2 \vee \ldots \vee q_m$ then Anke can choose one of these states and put it into P'. If there is exactly one successor state (no choice), Anke puts this one into P'. The successor state for the full set P is then the list of states which Anke has put into P'. This defines a nondeterministic run and that is successful iff after processing the word all members of the state of the nfa are accepting states of the afa. This permits to state the below theorem.

Theorem 4.17. *If an afa with n states recognises a language L then there is an nfa with up to 2^n states which recognises L.*

For Example 4.16 from above, the initial state would be $\{p\}$. If a 0 comes up, the next state is $\{p, q, r\}$, if a 1 comes up, the next state would be either $\{q\}$ or $\{r\}$. In the case that the nfa is in $\{p, q, r\}$ and a 0 comes up, the nfa remains in $\{p, q, r\}$. If a 1 comes up, all the states are mapped to $\{q, r\}$ and the state is accepting. Furthermore, in the case that the state is $\{q\}$, the successor chosen on 1 is $\{r\}$; in the case that the state is $\{r\}$, the successor chosen on 1 is $\{q\}$, in the case that the state is $\{q, r\}$, the successor chosen on 1 is $\{q, r\}$ again. These nondeterministic choices guarantee that the nfa accepts whenever the run ends with a 1 and one could eliminate other possible choices in order to obtain a dfa recognising the language $\{0, 1\}^* \cdot 1$. Indeed, there is even a two-state dfa doing the same.

One might ask in general how much the blow-up is when translating an afa into a dfa. This blow-up is double-exponential although it is slightly smaller than 2^{2^n}. In fact, using Büchi's construction to make the nfa from above a dfa would then lead to a dfa whose states are sets of subsets of Q. These sets, however, do not need to contain two subsets $A, B \subseteq Q$ with $A \subset B$. The reason is that any future set of states derived from B contains as a subset a future set of states derived from A and when all those of B are accepting, the same is true for those in A. Thus, it is safe to drop from a state $P \in Powerset(Powerset(Q))$ all those members $B \in P$ for which there is an $A \in P$ with $A \subset B$. Furthermore, the afa is defined above such that it has to be complete, so in both exponentiations the empty set is not used. Thus the cardinality of the so remaining states is smaller than 2^{2^n}. The next example, however, shows that the blow-up is still very large.

Example 4.18. *There is an afa with $2n + 2$ states such that the corresponding dfa has at least 2^{2^n} states.*

Proof. The alternating finite automaton has the states s, q and $p_1, \ldots, p_n$ and $r_1, \ldots, r_n$. The alphabet is $\{0, 1, \ldots, n\}$ (all considered as one-symbol digits). The transitions are given by the following table, where $i \in \{1, \ldots, n\}$ refers in the first two rows to some arbitrary value and in the last two rows to the index of p_i and r_i, respectively.

state	0	i	$j \in \{1, \ldots, n\} - \{i\}$
s	$s \vee q$	s	s
q	$p_1 \wedge \ldots \wedge p_n$	q	q
p_i	p_i	r_i	p_i
r_i	r_i	p_i	r_i

Let L be the language recognised by the afa. It contains all words u of the form $x0y0z$ such that $x, y, z \in \{0, 1, \ldots, n\}^*$ and z contains each of the digits $1, \ldots, n$ an even number of times.

Now consider for each subset $R \subset \{1, \ldots, n\}$ the word v_R consisting of the digits occurring in R taken once and $w_R = v_R 0 v_R$. Let S be some non-empty set of such sets R and R' be a member of S and u be the concatenation of $00v_{R'}$ with all w_R such that $R \in S$. For $R \subseteq \{1, \ldots, n\}$, the following statements are equivalent:

- $v_R \in L_u$;
- $uv_R \in L$;
- there is a 0 in uv_R such that all non-zero digits after this 0 appear an even number of times;
- either $v_R 0 v_R$ belongs to the components from which u is built or $R = R'$;
- $R \in S$.

Furthermore, one can see that L_ε does not contain any v_R. Thus for each $S \subseteq Powerset(\{1, \ldots, n\})$ there is an u with $R \in S \Leftrightarrow v_R \in L_u$ for all $R \in S$ and so there are 2^{2^n} many different derivatives. Any dfa recognising L must have at least 2^{2^n} states.

Indeed, one can make a dfa with $2^{2^n} + 1$ states: It has a starting state s which it only leaves on 0. All other states are members of $Powerset(Powerset(\{1, \ldots, n\}))$. On a 0, the state s is mapped to $\emptyset$. On each further 0, the state P is mapped to $P \cup \{\emptyset\}$. On symbol $k > 0$, a state P is mapped to $\{A \cup \{k\} : A \in P \wedge k \notin A\} \cup \{A - \{k\} : A \in P \wedge k \in A\}$. A state P is accepting iff P is different from s and P is a set containing $\emptyset$ as an element. ∎

If one scales the n in the above construction such that it denotes the number of states, then the theorem says that given an afa with n states, it might be that the corresponding dfa has at least $2^{2^{n/2-2}}$ states. This is near to the theoretical upper bound 2^{2^n} which, as said, is not the optimal upper bound. So the real upper bound is between $2^{2^{n/2-2}}$ and 2^{2^n}.

Exercise 4.19. *Show that every afa with two states is equivalent to a dfa with up to four states. Furthermore, give an afa with two states which is not equivalent to any dfa with three or less states.*

Theorem 4.20. *If there are n dfas $(Q_i, \Sigma, \delta_i, s_i, F_i)$ with m states each recognising $L_1, \ldots, L_n$, respectively, then there is an afa recognising $L_1 \cap \ldots \cap L_n$ with $1 + mn$ states.*

Proof. One assumes that the Q_i are pairwise disjoint; if they were not, this could be achieved by renaming. Furthermore, one chooses an additional starting state $s \notin \bigcup_i Q_i$ and let $Q = \{s\} \cup \bigcup_i Q_i$.

On symbol a, let $s \to \delta_1(s_1, a) \wedge \ldots \wedge \delta_n(s_n, a)$; furthermore, for all Q_i and $q_i \in Q_i$, on a let $q_i \to \delta_i(q_i, a)$. In other words, in the first step Boris chooses which of the automata for the L_i he wants to track down and from then on the automaton tracks exactly this automaton; Boris can win iff the word on the input is not in the L_i chosen.

The state s is accepting iff $\varepsilon \in L_1 \cap \ldots \cap L_n$ and a state $q_i \in Q_i$ is accepting iff $q_i \in Q_i$. Thus, in the case that the word on the input is in all L_i, whatever Boris choses, Anke will win the game; in the case that the word on the input is not in some L_i, Boris can choose this L_i in the first step to track and from then onwards, the automaton will follow this language and eventually accept. ∎

Example 4.21. Let L_i contain the word with an even number of digit i and $\Sigma = \{0, 1, \ldots, n\}$, $n = 3$. Now $Q_i = \{s_i, t_i\}$, $F_i = \{s_i\}$ and if $i = j$ then $\delta_i(s_i, j) = t_i, \delta_i(t_i, j) = s_i$ else $\delta_i(s_i, j) = s_i, \delta_i(t_i, j) = t_i$.

Now $Q = \{s, s_1, s_2, s_3, t_1, t_2, t_3\}$, on 0, $s \to s_1 \wedge s_2 \wedge s_3$, on 1, $s \to t_1 \wedge s_2 \wedge s_3$, on 2, $s \to s_1 \wedge t_2 \wedge s_3$, on 3, $s \to s_1 \wedge s_2 \wedge t_3$. On j, $s_i \to \delta_i(s_i, j)$ and $t_i \to \delta_i(t_i, j)$. The states s, s_1, s_2, s_3 are accepting.

The automaton does the following on the word 2021: First, on 2, the automaton transits by $s \to s_1 \wedge t_2 \wedge s_3$; then, on 0, the automaton updates $s_1 \wedge t_2 \wedge s_3 \to s_1 \wedge t_2 \wedge s_3$; then, on 2, it updates $s_1 \wedge t_2 \wedge s_3 \to s_1 \wedge s_2 \wedge s_3$; lastly, on 1, it updates $s_1 \wedge s_2 \wedge s_3 \to t_1 \wedge s_2 \wedge s_3$.

Note that one can write the states of the dfa equivalent to a given afa as a formula of alternatingly "and" and "or" between the afa states; then, when transiting on a, one replaces the afa states in the leaves by the corresponding formula on the right side of the arrow; at the end, when all input symbols are processed, one replaces all accepting afa states by the logical constant "true" and all rejecting afa states by the logical constant "false" and then evaluates the formula. So the above formula evaluates to "false" as t_1 is a rejecting states and it only contains conjunctions.

Exercise 4.22. *If there are n nfas $(Q_i, \Sigma, \delta_i, s_i, F_i)$ with m states each recognising $L_1, \ldots, L_n$, respectively, show that there is an afa recognising $L_1 \cap \ldots \cap L_n$ with $1 + (m + |\Sigma|) \cdot n$ states.*

In particular, for $n = 2$ and $\Sigma = \{0, 1, 2\}$, construct explicitly nfas and the product afa where L_1 is the language of all words where the last letter has already appeared before and L_2 is the language of all words where at least one letter appears an odd number of times.

The proof can be done by adapting the one of Theorem 4.20 where one has to replace dfas by nfas. The main adjustment is that, in the first step, one goes to new, conjunctively connected states which have to memorise the character just seen, as they can not yet do the disjunction of the nfa. The next step has therefore to take care of the disjunctions of two steps of the nfa, the one of the memorised first step plus the one of the next step. From then on, all rules are now disjunctive and not deterministic as before.

Exercise 4.23. *Assume that a game has an infinite board $\mathbb{N}$ and starts with three* *numbers a, b, c such that $a < b < c$; the initial value is $a = 12, b = 13, c = 14$. Possible moves are to increment one of the numbers by 1, as long as the condition on the order of the numbersis not violated. The game ends with a winner, when c becomesthe double of a. Anke starts to move. Is this game a winning game for Anke, a winning game for Boris or a draw game. Provide the winning strategy of the respective player or the draw strategies for both players.*

Exercise 4.24. *A game has fields $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$ and two markers which initially stand on 0 and 6. The players move alternately one of the markers by adding, modulo 12, either 1 or 2 to its position. When a player makes a move such that both markers stand on the same field, the game ends. Is this game a winning game for Anke, a winning game for Boris or a draw game. Provide the winning strategy of the respective player or the draw strategies for both players.*

Exercise 4.25. *Find a regular language L and a number n such that both the best dfa and nfa have n states but some afa needs less states.*

Exercise 4.26. *Construct an afa for the language of all decimal numbers which are not divisible by any 1-digit prime number.*

Exercise 4.27. *Consider a game on decimal numbers $a_n a_{n-1} \ldots a_1 a_0$. Players Anke and Boris move alternately. Determine for the below games which player wins from the following start situations: 300, 288, 1111, 1024. The player who reaches 0 wins. Let x denote the current number when the move is to be made, for each nonzero x, some move have to be made. Here the rule to move for players is as follows: The player can replace x by $x - y$ where y is odd and $y \leq x$.*

Exercise 4.28. *Do the same as in Exercise 4.27 with the only difference that the move-rule is now as follows: The player who moves has to reduce one non-zero digit by 1.*

Exercise 4.29. *Do the same as in Exercise 4.27 with the only difference that the move-rule is now as follows: The player who moves replaces one digit by a digit which is one or two or three smaller.*

Exercise 4.30. *Do the same as in Exercise 4.27 with the only difference that the move-rule is now as follows: The player can reduce one nonzero digit a_m by 1 and change (optional) one digit a_k with $k < m$ to an arbitrary value from $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$.*

Description 4.31. Consider the following game G on binary numbers. The "board" of the game is one binary number x. The players move alternatingly. In a move, the player makes the number smaller by either changing a 1 to a 0 or by interchanging a 1 with a more behind 0. The game terminates when 0 is reached and the player reaching 0 wins.

Exercise 4.32. *Determine which of the given binary numbers are winning for Anke in the game G from Description 4.31. Sketch the winning strategies for the winner to win the games. The numbers are* 1010, 10101010, 10000, 100001, 1111*.*

Exercise 4.33. *Determine which of the given binary numbers are winning for Anke in the game G from Description 4.31. Sketch the winning strategies for the winner to win the games. The numbers are* 1110, 111100, 110011001100, 101111, 101010, 10101000*.*

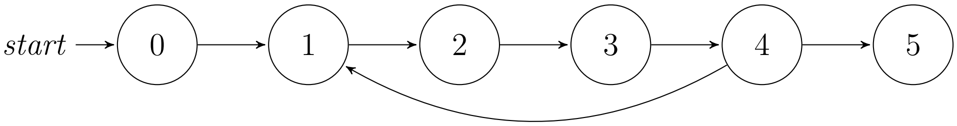
Exercise 4.34. *For the game G from Description 4.31, provide a regular infinite set of binary numbers such that each of them is a winning position for Boris (the player who moves second). Prove that this set works.*

5 Games Played for an Infinite Time

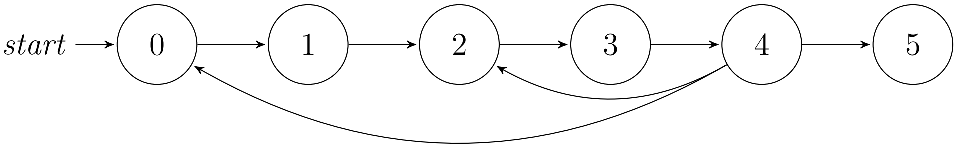
Infinite games are games where plays can go for an infinite time and nevertheless been won. A bit, the above finite games touched already this case, as for some case the draw was obtained by playing forever without going into a winning node for either player. The first type of game is a parallel situation: Anke wins iff the game runs forever.

Description 5.1: Survival Games. *Mathematicians consider time often as an infinite number of steps numbered as* $0, 1, 2, \ldots$*; so while there is a first time* 0*, there is no last time and the game runs forever. Nevertheless, there is an evaluation of the overall play of the two players. Such games might be still of some interest. The most easy of these is the survival game: One player (representing a human) can influence the environment by going from one state to another, the other player (nature or weather) can modify the environment in its own way and react to the human. Both player move alternating and the Human wins if he can avoid the bad nodes (representing unacceptable environment conditions) all the time. One can represent the bad nodes by nodes without outgoing edge; then the goal of the first player (Anke) would be to be able to move as long as possible while the goal of the second player (Boris) would be that the game after some time ends up in a node without outgoing edge.*

Example 5.2. *Anke and Boris move alternately in the following game. Anke starts in node* 0*. The game has a winning strategy for Anke, as it will always be her turn to move when the game is in node* 4 *and Boris cannot force the game to move into the dead end* 5*. The reason is that Anke will always move on nodes with even numbers and Boris on nodes with odd numbers.*



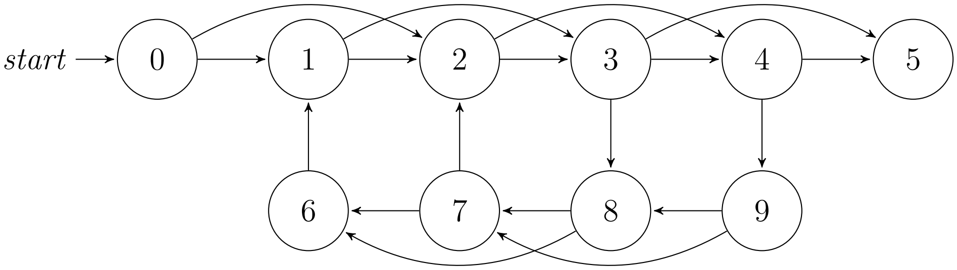
The next game is a slight modification of the above. Here now, when Anke wants is in node 4*, she can only move back to* 0 *or* 2 *so that Boris gets to move on even numbers while she will end up on moving on odd numbers. So the next time Boris has the choice to move on node* 4 *and can terminate the game by moving into* 5*.*



If one writes the possible plays with Boris moving into 5 *whenever possible, then the plays which arise in a way consistent with this strategy are* $0-1-2-3-4-5$ *(where Anke gives up),* $0-1-2-3-4-0-1-2-3-4-5$ *(where Anke moves from* 4 *to* 0 *and then Boris moves from* 4 *to* 5 *at the next time),* $0-1-2-3-4-2-3-4-5$ *(where Anke moves from* 4 *to* 2 *and then Boris moves form* 4 *to* 5 *at the next time).*

The winning strategy of Boris is memoryless: That is, whenever Boris makes a move, he does not have to consider the past, he has only to use the information in which node the game currently is. One can show that for a survival game, either Anke or Boris have always a memoryless winning strategy.

Quiz 5.3. *Consider the following game. Here Boris wins if the player reaches node* 5 *and Anke wins if the game never reaches this node.*



Which player has a winning strategy? Give a memoryless winning strategy for the player.

Theorem 5.4. *There is an algorithm which can check which player wins a survival game (when playing optimally). The algorithm runs in polynomial time.*

Proof. Let V be the set of vertices of the graph. Now one constructs a function f with domain $V \times \{\text{Anke}, \text{Boris}\}$ which tells for every node and every player whether the game will be lost for Anke within a certain amount of moves.

Make the partial function f as follows: $f(v, \text{Anke}) = 0$ or $f(v, \text{Boris}) = 0$ if the node v has no outgoing edge. Having once defined this, one extends the definition in rounds $n = 1, 2, \ldots, 2 \cdot |V|$ as follows: For each $v \in V$, if the value $f(v, \text{Anke})$ is still undefined and every outgoing edge $v \to w$ satisfies that $f(w, \text{Boris}) < n$ then let $f(v, \text{Anke}) = n$; if the value $f(v, \text{Boris})$ is still undefined and there is an outgoing edge $v \to w$ with $f(w, \text{Anke}) < n$ then let $f(v, \text{Boris}) = n$.

After $2 * |V|$ rounds, all values of f which can be defined in this way are defined, so further rounds would not add further values. Therefore, one now says that $f(v, \text{player}) = \infty$ for the remaining, not yet defined entries.

Now it is shown that the function f can be used to implement a memoryless winning strategy for that player who can win the game.

Let s be the starting node. If $f(s, \text{Anke}) = \infty$ then Anke has a winning strategy.

Each time, when its Anke's turn to move, she moves from the current node v with $f(v, \text{Anke}) = \infty$ to a node w with $f(w, \text{Boris}) = \infty$; if such a w would not exist then

$$f(v, \text{Anke}) < \max\{1 + f(w, \text{Boris}) : v \rightarrow w \text{ is an edge in the graph.}\} < \infty$$

in contradiction to the assumption on $f(v, \text{Anke})$. Now, Boris cannot move from w to any node u where $f(u, \text{Anke}) < \infty$, hence Boris moves to a node u with $f(u, \text{Anke}) = \infty$. So Anke can play in a way that the f remains on the value ∞ and will not end up in a node without outgoing edge. This strategy is obviously memoryless.

In the case that $f(s, \text{Anke}) < \infty$, Boris could play the game in a way that it takes at most $f(s, \text{Anke})$ moves. When the game is in a node with f taking the value 0, it has terminated; so consider the case that the value is larger than 0. If it is Anke's turn, she can only move from a node v to a node w with $f(w, \text{Boris}) < f(v, \text{Anke})$. If it is Boris' turn and $0 < f(v, \text{Boris}) < \infty$ then he can move to a node w with $f(w, \text{Anke}) < f(v, \text{Boris})$, so again the f value goes down. Also this strategy for Boris is obviously memoryless.

It is easy to see that the algorithm goes only through $2 * |V|$ rounds and in each round checks for $2 * |V|$ entries whether a certain condition is satisfied; this condition needs to follow all edges originating from v; therefore the overall algorithm is in $O(|V|^3)$, hence polynomial time. Note that this algorithm is not optimised for its runtime and that the cubic bound is not optimal; it is a special case of the algorithm in Theorem 4.6. ▮

Exercise 5.5. *Consider the following game $G(p, q, u, v, w)$ which is played on a graph G with u, v, w being vertices and $p \in \{Anke, Boris\}$ and $q \subseteq \{Anke, Boris\}$. Anke wins a play in this game if the game starts in node u and player p starts to move and the player moves alternately and the game goes through node v at some time and the game ends after finitely many steps in w with a player in the set q being the next to move.*

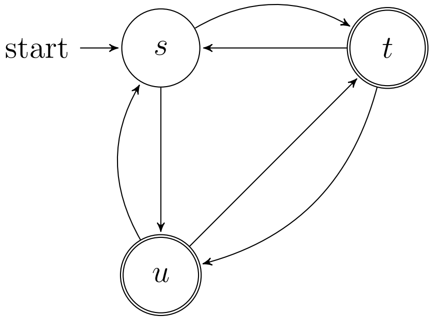
Note that if $q = \{Anke, Boris\}$ then the last condition on the last player to move is void. Furthermore, going through v includes the possibility that v is the first or last node to be visited so that the constraint on v is void in the case that $v = u$ or $v = w$. Furthermore, the graph might have more nodes than u, v, w; u, v, w are just the nodes mentioned as parameters of the game.

Give an algorithm to determine which player in this game has a winning strategy.

Description 5.6: Update Games. An update game is given by a graph with vertices V and edges E and a set W of special nodes such that Anke wins a game iff she visits during the infinite play each node in W infinitely often. Update games are so a special form of survival games, as they do not only require Anke to survive forever, but also to visit the nodes in W infinitely often; if the game gets stuck somewhere or runs forever without visiting every node in W infinitely often, then Boris wins.

Such a game might, for example, model a maintenance task where the maintenance people have to visit various positions regularly in order to check that they are in order and where various constraints — moves by own choice (player Anke) and moves imposed by other conditions beyond their control (player Boris) — influence how they navigate through this graph.

Example 5.7. Update games do not necessarily have memoryless strategies, as the following example shows (where the nodes in W are those with double boundaries).



Anke starts the game in s; whenever she moves to t or u, Boris moves the game back into s. Now, if Anke would have a memoryless strategy, she would always move to one of the two nodes in W, say to t, but then the other node in W, here u, will never be visited. So Anke has no memoryless winning strategy.

She has, however, a winning strategy using memory. Anke moves from node t to u, from node u to t and from node s to that node of u and t which has longer not yet been visited. So if Anke has to move in node s, she remembers from which node the game came to s. If the previous move (of Boris) was $t \to s$ then Anke moves $s \to u$ else Anke moves $s \to t$. This makes sure that after each visit of u, the node t is visited within 2 moves; furthermore, after each visit of t, the node u is visited within 2 moves.

Theorem 5.8. *There is an algorithm which determines which player has a winning strategy for an update game.*

Proof. Let s be the starting node and $w_1, w_2, \ldots, w_n$ be the nodes in W.

Now one first decides the following games $G(p, q, u, v, w)$ where $u, v, w \in V$ and $p \in \{\text{Anke}, \text{Boris}\}$ and q is a non-empty subsets of $\{\text{Anke}, \text{Boris}\}$. Now Anke wins the game $G(p, q, u, v, w)$, if it starts with some player p moving from u and it runs only finitely many steps visiting the node v in between and then ends up in w with a player in q being the next one to move. There are algorithms to decide this games, similar to those given in Theorem 4.6 and Theorem 5.4; Exercise 5.5 asks to design such an algorithm.

Now Anke has a winning strategy for the game iff one of the following conditions are satisfied:

- In Case 1 there is a node v and a player $p \in \{\text{Anke}, \text{Boris}\}$ such that Anke can win the game $G(\text{Anke}, \{p\}, s, v, v)$ and for each $w \in W$ Anke can win the game $G(p, \{p\}, v, w, v)$.
- In Case 2 there is a node v such that for every $w \in W$ and every player $p \in \{\text{Anke}, \text{Boris}\}$, Anke can win the games $G(\text{Anke}, \{\text{Anke}, \text{Boris}\}, s, v, v)$ and $G(p, \{\text{Anke}, \text{Boris}\}, v, w, v)$.

By assumption, this can be checked algorithmically. The algorithm is in polynomial time.

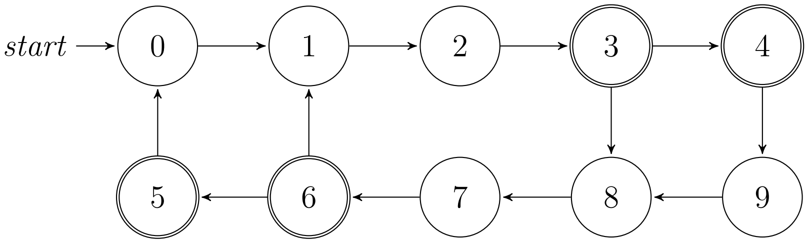
First it is verified that Anke has a winning strategy in the first case. She then can force from s that the game comes to node v and that it is player p to move. Furthermore, she can now alternatively for $w = w_1, w_2, \ldots, w_n$ force that the game visits this node and eventually returns to v with player p being the one to move. So she can force an infinite play which visits each of the nodes in W infinitely often.

Second it is verified that Anke has a winning strategy in Case 2. Anke can force the game into v without having a control which player will move onwards from v. Then, for each of the two cases, she can force that the game visits any given node $w \in W$ and returns to v, hence she can force that the game visits each of the nodes in W infinitely often.

Third assume that Case 1 and Case 2 both fail and that this is due because there is a player p and a node $u \in W$ such that Anke does not win $G(\text{Anke}, \{\text{Anke}, \text{Boris}\} - \{p\}, s, u, u)$ and Anke does not win $G(p, \{\text{Anke}, \text{Boris}\}, u, v, u)$ for some $v \in W$. Hence Boris can first enforce that either Anke visits u only finitely often or is at some point in u with the player to move being p; from then onwards Boris can enforce that the game either never reaches v or never returns to u after visiting v. Hence Boris has a winning strategy in this third case.

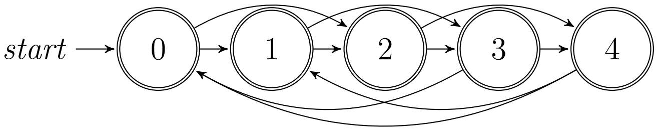
Fourth, assume that Cases 1 and 2 fail and that there are nodes $u, v, v' \in W$ such that Anke loses the games $G(\text{Anke}, \{\text{Anke}\}, u, v, u)$ and $G(\text{Boris}, \{\text{Boris}\}, u, v', u)$; hence Anke cannot enforce that a game visiting all nodes in W infinitely often has always the same player being on move when visiting u. As Case 2 fails, there is a node $w \in W$ and $p \in \{\text{Anke}, \text{Boris}\}$ such that Boris has a winning strategy for the game $G(p, \{\text{Anke}, \text{Boris}\}, u, w, u)$. It follows that once the player p is on the move in node u, Boris can enforce that either w or u is not visited again. Hence Boris can enforce that at least one of the nodes u, v, v', w is not visited infinitely often and so Boris has a winning strategy in this fourth case. This case distinction completes the proof. ▮

Quiz 5.9. *Which player has a winning strategy for the following update game?*



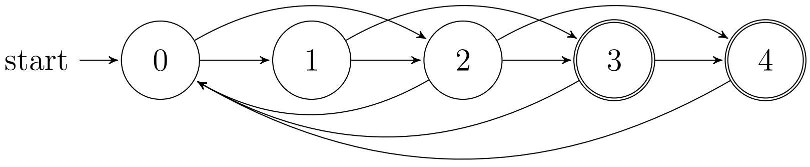
How much memory needs the strategy?

Exercise 5.10. *Let $n > 4$, n be odd, $V = \{m : 0 \leq m < n\}$ and $E = \{(m, m+1) : m < n-1\} \cup \{(m, m+2) : m < n-2\} \cup \{(n-2, 0), (n-1, 0), (n-1, 1)\}$ and $s = 0$. Show that Anke has a winning strategy for the update game (V, E, s, V) but she does not have a memoryless one.*



Here the game for $n = 5$.

Description 5.11. A Büchi game (V, E, s, W) is played on a finite graph with nodes V and edges E, starting node s and a special set $W \subseteq V$ (like an update game). Anke wins a play in the game iff, when starting in s, the game makes infinitely many moves and during these moves visits one or more nodes in W infinitely often.



Anke has a winning strategy for this game. The game is visiting node 0 infinitely often as all backward arrows end up in this node.

In the case that it is Anke's turn, she moves from 0 to 1. Then Boris can either move to 3 and visit one node in W or to 2 so that Anke in turn can move to 3 or 4 and hence visiting one of the accepting nodes.

In the case that it is Boris' move, he moves to 1 or 2 and Anke can go to the accepting node 3 from either of these nodes.

Theorem 5.12. *There is a polynomial time algorithm which decides which player can win a given Büchi game.*

Proof. Given a Büchi game (V, E, s, W), the idea is to make a function $f : V \times \{\text{Anke}, \text{Boris}\} \to \{0, 1, \ldots, 30 \cdot |V|^2\}$ which guides the winning strategies of Anke and Boris (which are memoryless).

Initialise $f(v, p) = 0$ for all nodes v and players p. The algorithm to compute f is to do the below updates as long as one of the if-conditions applies; if several apply, the algorithm does the statement of the first if-condition which applies:

- If there are nodes $v \in V - W$ and $w \in V$ with $(v, w) \in E$ and $f(v, \text{Anke}) < f(w, \text{Boris}) - 1$ then update $f(v, \text{Anke}) = f(v, \text{Anke}) + 1$;
- If there is $v \in V - W$ with an outgoing edge and all $w \in V$ with $(v, w) \in E$ satisfy $f(v, \text{Boris}) < f(w, \text{Anke}) - 1$ then update $f(v, \text{Boris}) = f(v, \text{Boris}) + 1$;
- If there are $v \in W$ with $f(v, \text{Anke}) \leq 30 \cdot |V|^2 - 3 \cdot |V|$ and $w \in V$ with $(v, w) \in E$ and $f(v, \text{Anke}) \leq f(w, \text{Boris}) + 6 \cdot |V|$ then update $f(v, \text{Anke}) = f(v, \text{Anke}) + 3 \cdot |V|$;
- If there is $v \in W$ with outgoing edge and $f(v, p) \leq 30 \cdot |V|^2 - 3 \cdot |V|$ and all $w \in V$ with $(v, w) \in E$ satisfy $f(v, \text{Boris}) \leq f(w, \text{Anke}) + 6 \cdot |V|$ then update $f(v, \text{Boris}) = f(v, \text{Boris}) + 3 \cdot |V|$.

Note that there are at most $60 \cdot |V|^3$ updates as each update increases one value of f by one and the domain has cardinality $2 \cdot |V|$ and the values in the range can be increased at most $30 \cdot |V|^2$ times. Hence the whole algorithm is polynomial in the size of $|V|$.

Now the strategy of Anke is to move from v to that node w with $(v, w) \in E$ for which $f(w, \text{Boris})$ is maximal; the strategy of Boris is to move from v to that node w with $(v, w) \in E$ for which $f(v, \text{Anke})$ is minimal.

As there are only $2 \cdot |V|$ many values in the range of f but the range can be spread out between 0 and $30 \cdot |V|^2$, there must, by the pigeon hole principle, be a natural number $m \leq 2 \cdot |V|$ such that there are no nodes v and players p with $10 \cdot |V| \cdot m \leq f(v, p) < 10 \cdot |V| \cdot (m + 1)$. Let m be the least such number. Now one shows the following claim.

Claim. If $f(v, p) \geq 10 \cdot m \cdot |V|$ then Anke can win the game else Boris can win the game.

To prove the claim, first observe that for all $w \in V - W$ with $f(w, \text{Anke}) > 0$ there is a node u with $(w, u) \in E$ and $f(u, \text{Boris}) = f(w, \text{Anke}) + 1$. The reason is that when $f(w, \text{Anke})$ was updated the last time, there was a successor $u \in V$ such that $f(u, \text{Boris}) > f(w, \text{Anke}) - 1$. Now this successor causes $f(w, \text{Anke})$ to be updated until $f(w,\text{Anke}) \geq f(u,\text{Boris})-1$. Similarly, for all $w \in V-W$ with $f(w,\text{Boris}) > 0$ one has that $f(w,\text{Boris}) = \min\{f(u,\text{Anke}) : (w,u) \in E\}-1$; again, this follows from the update rules. In particular as long as the game is in $V-W$, either the values of f remain constant at 0 or they go up. As there are only finitely many values, it means that the game eventually visits a node in W whenever the f-values of the current situation in the game is positive.

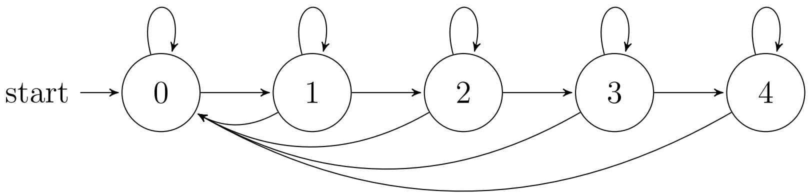
Now consider the case of any $w \in W$ and a player p moves from w to u following its strategy. If $p=$ Anke then $f(u,\text{Boris}) \geq f(w,\text{Anke}) - 6 \cdot |V|$; if $p=$ Boris then $f(u,\text{Anke}) \geq f(w,\text{Boris}) - 6 \cdot |V|$, as otherwise $f(w,p)$ would not have reached the final value in the update algorithm. Furthermore, unless $f(w,p) > 30 \cdot |V|^2 - 3 \cdot |V|$, one also can conclude that $f(w,p) \leq f(u,q) + 3 \cdot |V|$ for $q \neq p$, as otherwise a further update on $f(w,p)$ would have been done.

Thus, if $f(w,p) \geq 10 \cdot m \cdot |V|$ and the move from w to u is done in a play where Anke follows her winning strategy then actually $f(w,p) \geq 10 \cdot m \cdot |V| + 10 \cdot |V|$ and $f(u,q) \geq 10 \cdot m \cdot |V| + 10 \cdot |V|$ as well. Thus the game will go forever and visit nodes in W infinitely often.

However, consider now the case that the starting point satisfies $f(v,p) < 10 \cdot m \cdot |V|$. Then in particular $f(v,p) < 30 \cdot |V|^2 - 3 \cdot |V|$. For all (w,q) with $f(w,q) < 10 \cdot m \cdot |V|$ it holds that $f(w,q)$ cannot be increased as the conditions of the update-rules for f do no longer apply. This means that when $f(w,\text{Anke}) < 10 \cdot m \cdot |V|$ then all successor configurations satisfy the same condition; if $f(w,\text{Boris}) < 10 \cdot m \cdot |V|$ then some successor configuration satisfies the same condition. Furthermore, when $w \in W$ and $f(w,\text{Anke}) < 10 \cdot m \cdot |V|$ then all successor configurations (u,Boris) satisfy $f(u,\text{Boris}) < f(w,\text{Anke}) - 6 \cdot |V|$, if $f(w,\text{Boris}) < 10 \cdot m \cdot |V|$ and at least one successor configuration exists then there is a successor-node u with $f(u,\text{Anke}) < f(w,\text{Boris}) - 6 \cdot |V|$. Thus the play reduces each time by at least $3 \cdot |V|$ whenever it goes through a node in W. Furthermore, there is a constant c such that there is no w,p taking a value $f(w,p)$ having the remainder c modulo $3 \cdot |V|$; thus, the game cannot go up from a value of the form $3k \cdot |V|$ to $3(k+1) \cdot |V|$ without in between visiting a node in W. Therefore the game cannot visit a node in W infinitely often when starting at (v,p) with $f(v,p) \leq 10 \cdot m \cdot |V|$.

This two case distinctions complete the proof that the function f defines a memoryless winning strategy for one of the players in the given Büchi game. ∎

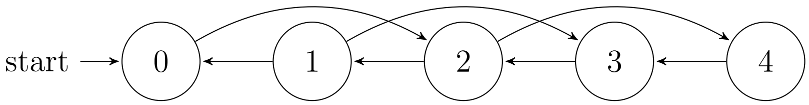
Description 5.13: Parity Games. A parity game is given by a graph (V,E) and a function *val* which attaches to each node $v \in V$ a value and a starting node $s \in V$. The players Anke and Boris move alternately in the graph with Anke moving first. Anke wins a play through nodes $v_0, v_1, \ldots$ in the game iff the limit superior of the sequence $val(v_0), val(v_1), \ldots$ is an even number.



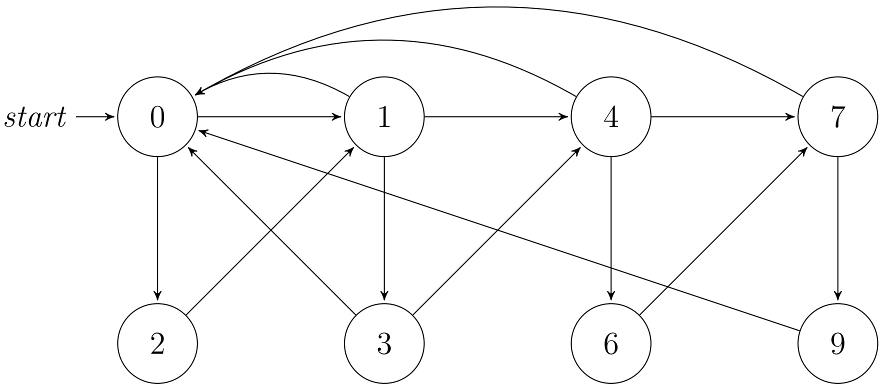
In this game, the nodes are labeled with their value, which is unique (what does not need to be). Anke has now the following memoryless winning strategy for this game: $0 \to 0$, $1 \to 2$, $2 \to 0$, $3 \to 4$, $4 \to 0$. Whenever the game leaves node 0 and Boris moves to node 1, then Anke will move to node 2. In the case that Boris moves the game into node 3, Anke will move to node 4. Hence whenever the game is in a node with odd value (what only happens after Boris moved it there), the game will in the next step go into a node with a higher even value. So the largest infinitely often visited node is even and hence the limit superior of this numbers is an even number. Hence Anke has a winning strategy for this parity game given here.

One can show that in general, whenever a player has a winning strategy for a parity game, then this winning strategy can be chosen to be memoryless; furthermore, there is always a player which has a winning strategy.

Quiz 5.14. *Which player wins this parity game? Give a winning strategy.*



Exercise 5.15. *Consider the following parity game:*



Which player has a winning strategy for this parity game? Give the winning strategy as a table (it is memoryless).

Description 5.16: Infinite games in general. The following general concept covers all the examples of games on finite graphs (V, E) with starting node s seen so far: The players Anke and Boris move alternately with Anke starting in s along the edges of the graph (which can go from a node to itself) and the winning condition consists of a function F from subsets of V to $\{\text{Anke}, \text{Boris}\}$ such that the winner of a play is $F(U)$ where U is the set of nodes visited infinitely often during the play. Here $U = \emptyset$ stands for the case that the game gets stuck after finitely many moves and no node is visited infinitely often. Here an overview how the winning conditions of the above games are translated into this general framework.

In the case of a survival game, $F(\emptyset) = \text{Boris}$ and $F(U) = \text{Anke}$ for all non-empty subsets $U \subseteq V$.

In the case of an update game with parameter W, if $W \subseteq U$ then $F(U) = \text{Anke}$ else $F(U) = \text{Boris}$.

In the case of a Büchi game with parameter W, if $W \cap U \neq \emptyset$ then $F(U) = \text{Anke}$ else $F(U) = \text{Boris}$.

In the case of a parity game, $F(\emptyset) = \text{Boris}$. For each non-empty set W, if $\max\{val(w) : w \in U\}$ is an even number (where the function val assigns to each node in V a natural number, see above) then $F(U) = \text{Anke}$ else $F(U) = \text{Boris}$.

There are games which can be captured by this framework and which are not of any of the types given above. For example, a game with $V = \{s, t, u\}$ where the players can move from each node to each node such that if $|U| = 2$ then $F(U) = \text{Anke}$ else $F(U) = \text{Boris}$.

Exercise 5.17. *Determine the function F for the following game: $V = \{0, 1, 2, 3, 4, 5\}$ the edges go from each node v to the nodes $(v + 1)$ mod 6 and $(v + 2)$ mod 6. The game starts in 0 and the players move alternately. Anke wins the game iff for each infinitely often visited node v, also the node $(v + 3)$ mod 6 is infinitely often visited.*

Define the function F on all possible values of U which can occur as an outcome of the game. List those values of U which cannot occur, that is, for which a value $F(U)$ does not need to be assigned. For example, as the game graph has for each node two outgoing edges, so it cannot get stuck somewhere and therefore $U = \emptyset$ is irrelevant.

Which player has a winning strategy for this game? Can this winning strategy be made memoryless?

Exercise 5.18. *Let a game (V, E, s, W) given by set V of nodes, possible moves E, starting node s and a set of nodes W to be avoided eventually. Let U be the infinitely often visited nodes of some play.*

Say that if $U \cap W = \emptyset$ and $U \neq \emptyset$ then Anke wins the game else Boris wins the *game.*

Determine an easy way mapping from (V, E, s, W) to (V', E', s', F') and players p to p' such that player p wins the avoidance game (V, E, s, W) iff p' wins the game (V', E', s', F') (see Description 5.16) where the type of (V', E', s', F') is one of survival game, update game, Büchi game or parity game. Say which type of game it is and how the mapping is done and give reasons why the connection holds.

Exercise 5.19. *Describe an algorithm which transforms a parity game (V, E, s, val) into a new parity game (V', E', s', val') such that this game never gets stuck and Anke wins (V, E, s, val) iff Boris wins (V', E', s', val'); without loss of generality it can be assumed that V is a finite subset of $\mathbb{N}$ and $val(v) = v$. Do the mapping such that V' has at most two nodes more than V.*

Exercise 5.20. *Consider the following game on the board $\mathbb{N}$: The game terminates at 1 and one can move from number $2n+2$ to $n+1$ and from number $2n+1$ to $(2n+1) \cdot k + h$ for some $k, h \in \{1, 3, 5, 7, 9\}$. Anke wins if the game runs forever and Boris wins if the game eventually reaches 1. Clearly Boris wins if the game starts in $2, 4, 8, 16$ and other powers of 2. Determine for all further numbers up to 20 who has a winning strategy when Anke starts at this number.*

Exercise 5.21. *Consider the game on the board $\mathbb{N}$ with the following moves: One can move from number $2n+2$ to $n+1$ and from number $2n+1$ to one of $2n+2$, $2n+4, 6n+6, 6n+8$. If the game reaches 1, Boris wins; if the game visits each node only finitely often without going to 1, Anke wins; if the game goes through some number infinitely often, it is draw. Give an algorithm which says which starting numbers are wins for Anke, wins for Boris and draw.*

Exercise 5.22. *Consider an update game on the board $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$ where players move by adding 1 or 3 modulo 12 to the current position and where Anke wins when $0, 4, 8$ are visited infinitely often. Which starting positions are wins for Anke and which are wins for Boris?*

Exercise 5.23. *Consider a Büchi game on $\{abcd : a, b, c, d \in \{0, 1, 2, 3\}\}$. One can add 1 or 2 modulo 4 to one out of four digits abcd. So from 0123, one can move to $1123, 2123, 0223, 0323, 0133, 0103, 0120, 0121$. Let $F = \{0000, 1111, 2222, 3333\}$. Determine the winner of the Büchi game where during the infinite duration of this game, one node in F has to be visited infinitely often.*

Exercise 5.24. *If one plays the game from Exercise 5.23 as a update game where each node in F has to be visited infinitely often and F is any nonempty set. Which player has a winning strategy?*

Exercise 5.25. *If one plays the Büchi game from Exercise 5.23 but with a modified set F which contains all nodes where three digits are equal, which player has a winning strategy?*

Exercise 5.26. *If one plays the Büchi game from Exercise 5.23 but with a modified set F which contains all nodes which have exactly one* 0*, which player has a winning strategy?*

Exercise 5.27. *Consider a game on $\{abcd : a, b, c, d \in \{0, 1, 2, 3\}\}$, where in a move one can add* 1 *or* 3 *modulo* 4 *to one out of four digits abcd. So from* 0123*, one can move to* 1123, 3123, 0223, 0023, 0133, 0113, 0120, 0122*. Let F contain all nodes which have at least two digits* 3*. Determine the winning positions of player Anke in a Büchi game and in the survival game. Recall that in a Büchi game, some node in F has to be visited infinitely often and that in a survival game, all the nodes in F have to be avoided forever. The survival game is lost for Anke if it starts in a node of F.*

Exercise 5.28. *Modify F from Exercise 5.27 such that F contains all nodes where at least one digit is* 3*. Determine the winning positions of the Büchi game and survival game for Anke in this game.*

6 Automata on Infinite Sequences

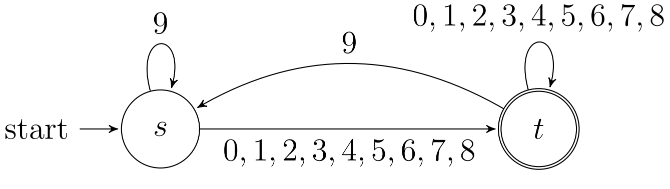
An infinite sequence $b_0b_1b_2\ldots \in \Sigma^\omega$ is a function from $\mathbb{N}$ to Σ; such sequences are called ω-words. One can for example represent all the real numbers between 0 and 1 by ω-words with $b_0b_1b_2\ldots \in \{0,1,2,3,4,5,6,7,8,8\}^\omega$ representing the sum $\sum_{k\in\mathbb{N}} 10^{-k}\cdot b_k$ where only the finite decimal fractions are not uniquely represented and have two representatives; for example $\frac{256}{1000}$ is represented by $25600000\ldots$ and $255999999\ldots$ while $\frac{1}{3}$ has the unique representative $333\ldots$ in this set. Representing real numbers is indeed one of the motivations in the study of ω-words and one can make more advanced systems which represent all numbers in $\mathbb{R}$. Richard Büchi and Lawrence Landweber [9, 10] investigated methods to deal with such representations in a way similar to what can be done with sets of finite words.

Description 6.1: Büchi Automata [9, 10]**.** A Büchi automaton is a nondeterministic automaton (Q,Σ,δ,s,F) where Q is the set of states, Σ the finite alphabet used, δ a set of possible transitions $(p,a,q)\in Q\times\Sigma\times Q$ such that the automaton can on symbol a go from p to q, s the starting state and F a set of states. Given an infinite word $b_0b_1b_2\ldots\in\Sigma^\omega$, a run on this sequence is a sequence $q_0q_1q_2\ldots\in Q^\omega$ of states such that $q_0=s$ and $(q_k,b_k,q_{k+1})\in\delta$ for all $k\in\mathbb{N}$. Let

$$U=\{p\in Q:\exists^\infty k\,[q_k=p]\}$$

be the set of infinitely often visited states on this run. The run is accepting iff $U\cap F\neq\emptyset$. The Büchi automaton accepts an ω-word iff it has an accepting run on this ω-word, otherwise it rejects the ω-word.

A Büchi automaton is called deterministic iff for every $p\in Q$ and $a\in\Sigma$ there is at most one $q\in Q$ with $(p,a,q)\in\delta$; in this case one also writes $\delta(p,a)=q$. The following deterministic Büchi automaton accepts all decimal sequences which do not have an infinite period of 9; that is, each real number r with $0\leq r<1$ has a unique representation in the set of sequences accepted by this automaton.



This automaton goes infinitely often through the accepting state t iff there is infinitely often one of the digits $0,1,2,3,4,5,6,7,8$ and therefore the word is not of the form $w9^\omega$.

Exercise 6.2. *Make a deterministic Büchi automaton which accepts an ω-word from $\{0,1,2\}^\omega$ iff it contains every digit infinitely often.*

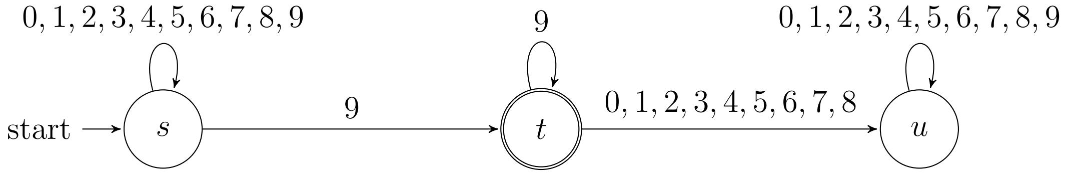
Exercise 6.3. *Make a deterministic Büchi automaton which accepts an ω-word from $\{0,1,2\}^\omega$ iff it contains at least two digits infinitely often.*

Exercise 6.4. *Make a deterministic Büchi automaton with three states which accepts all ω-words in which at least six of the usual ten decimal digits occur infinitely often and which rejects all ω-words where only one digit occurs infinitely often. There is no requirement what the automaton does on other ω-words.*

While nondeterministic and deterministic finite automata have the same power, this is not true for Büchi automata.

Example 6.5. Let L be the language of all ω-words which from some point onwards have only 9s, so $L = \{0,1,2,3,4,5,6,7,8,9\}^* \cdot 9^\omega$. Then there is a nondeterministic but no deterministic Büchi automaton accepting L. Furthermore, the languages recognised by deterministic Büchi automata are not closed under complement.

First, one shows that a nondeterministic Büchi automaton recognises this language. So an accepting run would at some time guess that from this point onwards only 9s are coming up and then stay in this state forever, unless some digit different from 9 comes up.



Second, it has to be shown that no deterministic Büchi automaton recognises this language. So assume by way of contradiction that $(Q, \{0,1,2,3,4,5,6,7,8,9\}, \delta, s, F)$ would be such an automaton. Now one searches inductively for strings of the form $\sigma_0, \sigma_1, \ldots \in 09^*$ such that $\delta(s, \sigma_0\sigma_1 \ldots \sigma_n) \in F$ for all n. If there is an n such that σ_n cannot be found then the sequence $\sigma_0\sigma_1 \ldots \sigma_{n-1}09^\omega$ is not accepted by the Büchi automaton although it is in L, as states in F are visited only finitely often in the run. If all σ_n are found, then the infinite sequence $\sigma_0\sigma_1 \ldots$ has the property that it contains infinitely many symbols different from 9 and that it is not in L; however, the automaton visits infinitely often a state from F. This contradiction shows that L cannot be recognised by a deterministic Büchi automaton. ▮

Quiz 6.6. *Show the following: if L and H are languages of ω-words recognised by deterministic Büchi automata then also $L \cup H$ and $L \cap H$ are recognised by deterministic Büchi automata. Show the same result for nondeterministic Büchi automata.*

Quiz 6.7. *Make a Büchi automaton which recognises the language of all ω-words in which exactly two digits from $\{0,1,2,3,4,5,6,7,8,9\}$ occur infinitely often. Describe how the automaton is build; it is not necessary to make a complete diagram of its states.*

Product Automata 6.8. Given ω-languages L and H, assume that the automata $(Q_L, \Sigma, \delta_L, s_L, F_L)$ and $(Q_H, \Sigma, \delta_H, s_H, F_H)$ recognise L and H, respectively.

Now let $Q = Q_L \times Q_H \times \{10, 01, 11\}$ and for each $(q_L, q_H) \in Q_L \times Q_H$ and $a \in \Sigma$ let $\delta((q_L, q_H, r), a) = (\delta_L(q_L, a), \delta_H(q_H, a), r')$ where $r' = r$ if $q_L \notin F_L$ and $q_H \notin F_H$ and $r' = F_L(q_L)F_H(q_H)$ if at least one of these bits is 1.

For the union, $(q_L, q_H, r) \in F$ if either $q_L \in F_L$ or $q_H \in F_H$. For the intersection, $(q_L, q_H, r) \in F$ iff $q_L \in F_L$ and the second bit or r is 1 or $q_H \in F_H$ and the first bit of r is 1.

Example 6.9. Let L is the set of all ω-words containing infinitely many even digits and let H is the set of all ω-words containing infinitely often either 0 or 5.

A Büchi automaton for L consists of two states s_L, t_L where the automaton goes to t_L iff it has just seen an even digit and to s_L iff it has just seen an odd digit. $F_L = \{t_L\}$.

A Büchi automaton for H consits of two states s_H, t_H where the automaton goes to t_H iff it has just seen 0 or 5 and to s_H otherwise. $F_H = \{t_H\}$.

The intersection $L \cap H$ contains all ω-words where infinitely many even digits and also infinitely many digits from $0, 5$ appear in the ω-word.

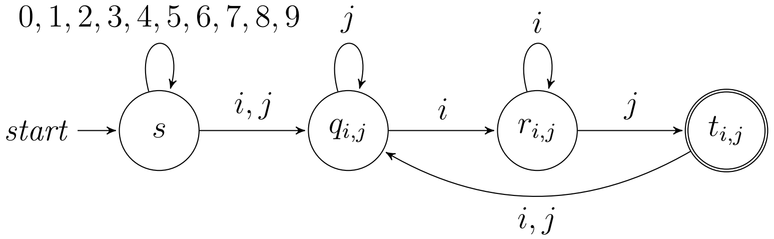
Product automaton has states from $\{s_L, t_L\} \times \{s_H, t_H\} \times \{01, 10, 11\}$.

The successor of (q_L, q_H, r) on input a is determined as follows: Let $r' = r$ for (s_L, s_H, r), $r' = 10$ for (t_L, s_H, r), $r' = 01$ for (s_L, t_H, r) and $r' = 11$ for (t_L, t_H, r). Now

$$\begin{aligned}
&\delta((q_L, q_H, r), 0) = (t_L, t_H, r');\\
&\delta((q_L, q_H, r), 5) = (s_L, t_H, r');\\
&\delta((q_L, q_H, r), a) = (t_L, s_H, r') \text{ for } a \in \{2,4,6,8\};\\
&\delta((q_L, q_H, r), a) = (s_L, s_H, r') \text{ for } a \in \{1,3,7,9\}.
\end{aligned}$$

Starting state is $(s_L, s_H, 11)$. The set F of accepting states contains all nodes of form $(t_L, q_H, x1)$ and of form $(q_L, t_H, 1x)$. Here q_L, q_H are any states in the corresponding automata and x is any bit 0 or 1.

Example 6.10. *Consider the following automaton $B_{i,j}$ with $i \neq j$.*



Now make a central starting node s connected to cycles of three nodes $q_{i,j}, r_{i,j}, t_{i,j}$ for all pairs of distinct digits i, j as in the diagramme above. The nodes $t_{i,j}$ are the only accepting ones.

Exercise 6.11. *Construct deterministic Büchi automata for the language L_{ab} of all ω-word which do not contain the subword ab anywhere. Then construct the intersection automaton for $L_{01} \cap L_{23}$. The alphabet is $\{0, 1, 2, 3\}$.*

Exercise 6.12. *Cnstruct a deterministic Büchi automaton for the language H_{ab} of all ω-words which in which the subword ab occurs infinitely often. Then construct the intersection via a product automaton for $H_{01} \cap H_{23}$. The alphabet is $\{0, 1, 2, 3\}$.*

Exercise 6.13. *Construct a deterministic Büchi automaton with four states for $H_{01} \cup H_{23}$ with H_{ab} as in Exercise 6.12. This automaton does not need to be of the form of a product automaton. The alphabet is $\{0, 1, 2, 3\}$.*

Büchi [9] found the following characterisation of the languages of ω-words recognised by nondeterministic Büchi automata.

Theorem 6.14: Büchi's Characterisation [9]**.** *The following are equivalent for a language L of ω-words:*
(a) *L is recognised by a nondeterministic Büchi automaton;*
(b) *$L = \bigcup_{m \in \{1,\ldots,n\}} A_m B_m^\omega$ for some n and $2n$ regular languages $A_1, B_1, \ldots, A_n, B_n$.*

Proof, (a)⇒(b). Assume that a nondeterministic Büchi automaton $(Q, \Sigma, \delta, s, F)$ recognises L. Let n be the cardinality of F and $p_1, p_2, \ldots, p_n$ be the states in F. Now let A_m be the language of strings recognised by $(Q, \Sigma, \delta, s, \{p_m\})$ when viewed as an nfa and B_m be the set of non-empty words in the language recognised by $(Q, \Sigma, \delta, p_m, \{p_m\})$ when viewed as an nfa. It is clear that all these languages are regular. Furthermore, for each ω-word in $A_m \cdot B_m^\omega$ there is a run of the original Büchi automaton which goes infinitely often through the state p_m. Furthermore, if a ω-word has a run which goes infinitely often through states in the set F then there is at least one state p_m which is infinitely often visited by this run; one can now partition this

ω-word in parts $\sigma_0, \sigma_1, \sigma_2, \ldots$ such that the run is after processing $\sigma_0\sigma_1 \ldots \sigma_k$ in p_m for visit number k (with the first visit having number 0). Note that all σ_k are non-empty and that $\sigma_0 \in A_m$ and each σ_k with $k > 0$ is in B_m. Hence the ω-word is in $A_m B_m^\omega$.

(b)⇒(a). Assume that $L = A_1 \cdot B_1^\omega \cup A_2 \cdot B_2^\omega \cup \ldots \cup A_m \cdot B_n^\omega$. Assume that each language A_m is recognised by the nfa $(N_{2m-1}, \Sigma, \delta_{2m-1}, s_{2m-1}, F_{2m-1})$ and each language $B_m \cup \{\varepsilon\}$ is recognised by the nfa $(N_{2m}, \Sigma, \delta_{2m}, s_{2m}, F_{2m})$; note that here $s_{2m} \in F_{2m}$ for all m.

Now let $N_0 = \{s_0\} \cup N_1 \cup N_2 \cup \ldots \cup N_{2n}$ where all these sets are considered to be disjoint (by renaming the non-terminals, if necessary). The start symbol of the new automaton is s_0. Furthermore, let δ_0 be $\delta_1 \cup \delta_2 \cup \ldots \cup \delta_{2n}$ plus the following transitions for each $a \in \Sigma$: first, (s_0, a, q) if there is an m such that $(s_{2m-1}, a, q) \in \delta_{2m-1}$; second, (s_0, a, q) if there is an m such that $\varepsilon \in A_m$ and $(s_{2m}, a, q) \in \delta_{2m}$; third, (s_0, a, s_{2m}) if $a \in A_m$; fourth, (q, a, s_{2m}), if there are m and $p \in F_{2m-1}$ with $q \in N_{2m-1}$ and $(q, a, p) \in \delta_{2m-1}$; fifth, (q, a, s_{2m}), if there are m and $p \in F_{2m}$ with $q \in N_{2m}$ and $(q, a, p) \in \delta_{2m}$. These added rules reconnect the transitions from the starting state into A_m (first case) or directly into B_M (when $\varepsilon \in A_m$ for the second case or $a \in A_m$ for the third case) and the transition from A_m into B_m on the reach of an accepting state of A_m (fourth case) and the return to the starting state of B_m when an accepting state can be reached (fifth case). In the fourth or fifth case, the automaton could go on in A_m or B_m instead of going to s_{2m}, this choice is left nondeterministic on purpose; indeed, one cannot in all cases make this Büchi automaton deterministic.

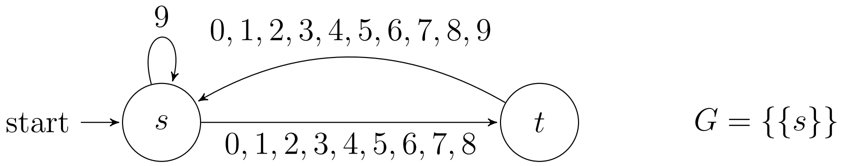
Now $\{s_2, s_4, s_6, \ldots, s_{2n}\}$ is the set F_0 of final states of the newly constructed Büchi automaton $(N_0, \Sigma, \delta_0, s_0, F_0)$. That is, this Büchi automaton accepts an ω-word iff there is a run of the automaton which goes infinitely often through a node of the form s_{2m}; by the way how δ_0 is defined, this is equivalent to saying that the given ω-word is in $A_m \cdot B_m^\omega$. The further verification of the construction is left to the reader. ▮

Muller introduced a notion of automata which overcomes the shortage of deterministic Büchi automata and can nevertheless be made deterministic.

Description 6.15: Muller automaton. A Muller automaton $(Q, \Sigma, \delta, s, G)$ consists of a set of states Q, an alphabet Σ, a transition relation δ, a starting state s and a set G of subsets of Q. A run of the Muller automaton on an ω-word $b_0 b_1 b_2 \ldots \in \Sigma^\omega$ is a sequence $q_0 q_1 q_2 \ldots$ with $q_0 = s$ and $(q_k, a, q_{k+1}) \in \delta$ for all k. A run of the Muller automaton is accepting iff the set U of infinitely often visited states satisfies $U \in G$. The Muller automaton accepts the ω-word $b_0 b_1 b_2 \ldots$ iff it has an accepting run on it.

A Muller automaton is deterministic iff the relation δ is a function, that is, for each $p \in Q$ and $a \in \Sigma$ there is at most one $q \in Q$ with $(p, a, q) \in \delta$.

While the language of all ω-words of the form $w9^\omega$ is not recognised by a deterministic Büchi automaton, it is recognised by the following deterministic Muller automaton: $(\{s,t\},\{0,1,2,3,4,5,6,7,8,9\},\delta,s,\{\{s\}\})$ where $\delta(s,a) = t$ for $t < 9$, $\delta(s,9) = s$ and $\delta(t,a) = s$. The following diagram illustrates the Muller automaton:



If the ω-word consists from some time onwards only of 9s then the automaton will after that time either go to s or be in s and then remain in s forever so that $U = \{s\}$ and $U \in G$; hence the run of the automaton on this ω-word is accepting.

If the ω-word consists of infinitely many symbols different from 9 then the automaton will leave on its run infinitely often the state s for the state t and $U = \{s,t\}$ what is not in G. Hence the run of the automaton on this ω-word is rejecting.

As the automaton is deterministic, the automaton accepts a word iff it contains from some point onwards only 9s.

Exercise 6.16. *Make a deterministic Muller automaton which accepts all those ω-words in which at least one of the symbols from the alphabet $\{0,1,2\}$ occurs only finitely often.*

Exercise 6.17. *Make a deterministic Muller automaton with alphabet $\{0,1,2\}$ which recognises the language of all ω-words for which there is an even number of $ab \in \{01,12,20\}$ which occurs infinitely often as a subword in the ω-word.*

Exercise 6.18. *Make a deterministic Muller automaton with alphabet $\{0,1,2\}$ which recognises the language of all ω-words for which there are no $ab \in \{21,10,02\}$ which occur infinitely often as a subword in the ω-word.*

McNaughton [62] established the equivalence of these three conditions. The direct translation from (a) to (b) was also optimised by Safra [76, 77].

Theorem 6.19: McNaughton's Characterisation [62]. *The following are equivalent for a language L of ω-words:*
(a) *L is recognised by a nondeterministic Büchi automaton;*
(b) *L is recognised by a deterministic Muller automaton;*
(c) *L is recognised by a nondeterministic Muller automaton.*

Proof, (a)⇒(b). Let a nondeterministic Büchi automaton (Q,Σ,δ,s,F) be given. Without loss of generality, the given Büchi automaton has on every ω-word some infinite run; if not, one could add one state q such that one can go from every state into q, $q \notin F$ and one can go from q only into q itself (irrespective of the input).

The idea is that the corresponding deterministic Muller automaton keeps track of which non-terminals can be reached and how the history of visiting accepting states before the current state is. So for each initial part of a nondeterministic run $q_0 q_1 q_2 \ldots q_n$ of the Büchi automaton while processing input $b_1 b_2 \ldots b_n$, the Muller automaton machine would ideally archive the current state q_n and a string $\tau \in \{0,1\}^{n+1}$ representing the history where $\tau(m) = 0$ for $q_m \in F$ and $\tau(m) = 1$ for $q_m \notin F$. So the overall goal would be to have as many 0 as possible in the string τ and to have them as early as possible (what might be even more important than the actual number).

Hence, if there are two different runs having the traces τ and η, both ending up in the same state q_n, then the learner would only archive $\min_{lex}\{\tau, \eta\}$. For that reason, after processing $b_1 b_2 \ldots b_n$, the Muller automaton would represent the current state by a set of pairs (p, τ) with p being a state reachable on input $b_1 b_2 \ldots b_n$ and τ being the lexicographically least string representing the history of some run on this input. Note that if one can go with histories τ, τ' from s to p on $b_1 b_2 \ldots b_n$ and with histories η, η' from p to q on $b_{n+1} b_{n_2} \ldots b_m$ then one can also go with each of the histories $\tau\eta$, $\tau\eta'$, $\tau'\eta$, $\tau'\eta'$ from s to q on $b_1 b_2 \ldots b_m$. Furthermore, the lexicographic minimum of these four strings is $\min_{lex}\{\tau, \tau'\} \cdot \min_{lex}\{\eta, \eta'\}$. For that reason, it is save always only to store the lexicographic minimum. Furthermore, for determining the lexicographic minimum for going from s to q on $b_1 b_2 \ldots b_m$, it is sufficient to know for each state p the lexicographic minimum of the history on the first n steps and then the history for the resulting m steps will be one of these strings in $\{0,1\}^{n+1}$.

The disadvantage of this approach is that the information archived is growing and growing, that is, in each step the strings archived in the pairs get longer and longer. Therefore, one does the more complicated following algorithm to update the state, which is represented by a set of pairs (p, τ) where all the strings in the state are of same length and where for each $p \in Q$ there is at most one pair (p, τ) in the state of the Muller automaton. Furthermore, one stores in the state besides these pairs one special symbol representing a number between 0 and the maximum length of a τ which represents a column deleted in the last step. For the next state, this number is irrelevant.

First one creates the initial state $\tilde{s}$ of the Muller automaton by taking $\{\infty, (s, 0)\}$ in the case that $s \in F$ and $\{\infty, (s, 1)\}$ in the case that $s \notin F$. Now, one determines for each state $\tilde{p}$ created so far and each symbol b a successor state $\delta(p)$ as follows; one keeps adding these successor states to the set P of states of the Muller automaton until no new state is created this way. Let a state $\tilde{p} \in P$ and a symbol $b \in \Sigma$ be given.

1. Let $Q' = \{q : \exists \sigma_q \, [(q, \sigma_q) \in \tilde{p}]\}$.
2. Now one determines whether there is a k such that for all σ_q, σ_p: The first bit differing in σ_p, σ_q is not at position k and if $\sigma_p(k) = 0$ then there is a $k' > k$

with $\sigma_p(k') = 0$ as well. If this k exists then choose the least among all possible values else let $k = \infty$.

3. Start creating the state $\Delta(\tilde{p}, b)$ by putting the element k into this state (which is considered to be different from all pairs).
4. Let τ_q be obtained from σ_q by omitting the bit at position q in the case $k < \infty$ and by letting $\tau_q = \sigma_q$ in the case $k = \infty$.
5. For each $q \in F$, determine the set $\{\tau_p 0 : p \in Q' \wedge (p, b, q) \in \delta\}$. If this set is not empty, then let η_q is lexicographic minimum and put (q, η_q) into the new state $\Delta(\tilde{p}, b)$.
6. For each $q \in Q - F$, determine the set $\{\tau_p 1 : p \in Q' \wedge (p, b, q) \in \delta\}$. If this set is not empty, then let η_q is lexicographic minimum and put (q, η_q) into the new state $\Delta(\tilde{p}, b)$.
7. The new state $\Delta(\tilde{p}, b)$ consists of all the information put into this set by the above algorithm.

Now one shows that whenever the σ_q in the state $\tilde{p}$ have at least length $2 * |Q|$ then $k < \infty$ and therefore the η_q in the state $\Delta(\tilde{p}, b)$ will have the same length as the σ_q, hence the length of the archived strings is not increasing and so the number of states created is finite, actually P has at most $2^{2*|Q|^2+|Q|} * (2 * |Q| + 1)^2$ members.

To see this, assume that all the strings σ_q in $\tilde{p}$ have length $2 * |Q|$. There are at most $|Q|$ of these strings. They and their prefixes form a binary tree with up to $|Q|$ leaves of length $2 * |Q|$ and so there are at most $|Q| - 1$ branching nodes σ' in the tree. For each branching node σ', there are $\sigma' 0, \sigma' 1$ in the tree. Now let $K = \{|\sigma'| : \sigma' \text{ is a branching node in the tree}\}$. Furthermore, for each leave σ_p, let k' be the largest value where $\sigma_p(k') = 0$; for each σ_p add this k' into K. Then K has at most $2 * |Q| - 1$ many members. Hence $k = \min(\{0, 1, \ldots, 2 * |Q| - 1\} - K)$ exists and is identical to the k chosen in step 2, as for all σ_p, σ_q, if $\sigma_p(k) = 0$ then some $\sigma_p(k') = 0$ for $k' > k$ and if $\sigma_p(k) \neq \sigma_q(k)$ then there is some $k' < k$ with $\sigma_p(k') \neq \sigma_q(k')$. Hence the τ_q are shorter than the σ_q by one bit and the η_q have the same length as the τ_q. Furthermore, it is clear that Δ is a function and the resulting Muller automaton will be deterministic.

The remaining part of the Muller automaton to be defined is G: So let G contain every set W such that there is a $k < \infty$ satisfying $k = \min\{k' : \exists \tilde{p} \in W\, [k' \in \tilde{p}]\}$ and $W \cap U_k \neq \emptyset$ where

$$U_k = \{\tilde{p} : \exists \vartheta \in \{0,1\}^k\, [\exists (q, \sigma) \in \tilde{p}\, [\sigma \in \vartheta \cdot \{0,1\}^*] \wedge \forall (q, \sigma) \in \tilde{p}\, [\sigma \notin \vartheta \cdot 1^*]]\}.$$

Consider any given ω-word and let W be the set of states which the Muller automaton visits on this ω-word infinitely often. Let $\tilde{p}_m$ denote $\Delta(\tilde{s}, b_1 b_2 \ldots b_m)$. There is an n so large that $\tilde{p}_m \in W$ for all $m \geq n$.

First assume that there is an $\tilde{p}_{n'} \in U_k$ for some $n' > n$. Now the Muller automaton accepts $b_1 b_2 \ldots$, as $p_{n'} \in W$. So one has to show that $b_1 b_2 \ldots$ is also accepted by the Büchi automaton. As $\tilde{p}_{n'} \in U_k$ there is a $\vartheta \in \{0,1\}^k$ satisfying $\exists (q,\sigma) \in \tilde{p}_{n'} \, [\sigma \in \vartheta \cdot \{0,1\}^*] \wedge \forall (q,\sigma) \in \tilde{p}_{n'} \, [\sigma \notin \vartheta \cdot 1^*]$. Now consider $\Delta(\tilde{p}_{n'}, b_{n'})$. By construction, whenever $(q,\sigma) \in \tilde{p}_{n'}$ and σ extends ϑ then $\sigma(k') = 0$ for some $k' \geq k$; this property is preserved by the corresponding τ associated with q in $\tilde{p}_{n'}$. Furthermore, each pair $(q',\sigma) \in \tilde{p}_{n'+1}$ satisfies that $\sigma = \tau a$ for some a and the lexicographically least τ which belongs to some $q \in Q$ with $(q, b_{n'}, q') \in \delta$; whenever that τ extends ϑ then it contains a 0 after position k and therefore η has the same property. Hence every pair $(q',\sigma) \in \tilde{p}_{n'+1}$ satisfies that either σ does not extend ϑ or σ extends ϑ with a string containing a 0. Furthermore, if some of the $\tilde{p}_m$ with $m > n'$ would not contain any (q,σ_q) with σ_q extending ϑ, then this property would inherit to all $\tilde{p}_o$ with $o > m$; as $\tilde{p}_{n'} = \tilde{p}_o$ for infinitely many o, this cannot happen. Hence all members of W are in U_k as witnessed by the same ϑ.

Now let T be the tree of all finite runs $q_0 q_1 \ldots q_m$ such that there is an associated sequence $(\sigma_0, \sigma_1, \ldots, \sigma_m)$ of strings with $(q_h, \sigma_h) \in \tilde{p}_h$ for all $h \leq m$ and $\vartheta \preceq \sigma_h$ for all h with $n \leq h \leq m$ and satisfying for all $h < m$ and the $k' \in \tilde{p}_{h+1}$ that $(q_h, b_h, q_{h+1}) \in \delta$ and σ_{h+1} is obtained from σ_h by omitting the k'-th bit (in the case that $k' \neq \infty$) and then appending 0 in the case that $q_{h+1} \in F$ and appending 1 in the case that $q_{h+1} \notin F$. Each pair in each $\tilde{p}_m$ for each m must be reachable by such a sequence; hence T is infinite. By König's Lemma there is an infinite sequence of q_m of states with the corresponding sequence of σ_m with the same property. This sequence then satisfies that from some n onwards there is always a 0 somewhere after the k-th bit in σ_m; furthermore, the k-th bit is infinitely often deleted; hence it is needed that infinitely often a 0 gets appended and so the sequence $q_0 q_1 \ldots$ satisfies that $q_m \in F$ for infinitely many m. Hence $b_0 b_1 \ldots$ has the accepting run $q_0 q_1 \ldots$ of the Büchi automaton.

Second assume that there is no $\tilde{p}_m \in U_k$ for any $m \geq n$. Then the Muller automaton rejects the run and one has to show that the Büchi automaton does the same. Assume by way of contradiction that there is an accepting run $q_0 q_1 q_2 \ldots$ of the Büchi automaton on this sequence and let $\sigma_0, \sigma_1, \sigma_2, \ldots$ be the corresponding strings such that $(q_m, \sigma_m) \in \tilde{p}_m$ for all m. There is a string $\vartheta \in \{0,1\}^k$ which is for almost all m a prefix of the σ_m. Furthermore, by assumption, there is for all $m \geq n$ a state $r_m \in Q$ with $(r_m, \eta 1^\ell) \in \tilde{p}_m$. There are infinitely many m with $\sigma_m \succeq \vartheta \wedge \sigma_m \neq \vartheta 1^\ell$. Let k' be the minimum of the $k' \geq k$ such that $\sigma_{n''}(k') = 0 \wedge \vartheta \preceq \sigma_{n''}$ for some $m \geq n$; the minimal k' exists; fix some n'' with the chosen property. Then the number $k'' \in \tilde{p}_{n''+1}$ satisfies that $k'' \geq k$. Furthermore, $k'' \neq k'$ as $(r_{n''}, \vartheta 1^\ell)$ and $(q_{n''}, \sigma_{n''})$ are both in $\tilde{p}_{n''}$ and k' is the first position where they differ. Furthermore, it does not happen that $k \leq k'' < k'$ as then $\sigma_{n''+1}$ would have the 0 at $k'-1$ and $k'-1 \geq k$ in contradiction to the choice of k'. Hence not k but $k'+1$ would be the limit inferior of all the numbers in $\tilde{p}_m$ with $m \geq n''$ which equals to the states in W, a contradiction. Thus such an accepting run of the Büchi automaton on $b_0 b_1 b_2 \ldots$ does not exists and the

Büchi automaton rejects the input in the same way as the Muller automaton does.

So it follows from the case distinction that both automata recognise the same language. Furthermore, the deterministic Muller automaton constructed from the Büchi automaton has exponentially many states in the number of states of the original nondeterministic Büchi automaton.

(b)⇒(c). This holds, as every deterministic Muller automaton can by definition also be viewed as a nondeterministic one.

(c)⇒(a). Let a nondeterministic Muller automaton $(Q, \Sigma, \delta, s, G)$ be given. Let $succ(w, W)$ be a function which cycles through the set W: if $W = \{w_1, w_2, w_3\}$ then $succ(w_1, W) = w_2$, $succ(w_2, W) = w_3$ and $succ(w_3, W) = w_1$. Now let $P = Q \times Q \times (G \cup \{\emptyset\})$ is the set of states of the equivalent Büchi automaton, the alphabet Σ is unchanged, the starting state is $(s, s, \emptyset)$ and for each transition (p, a, q) and each r and each $W \in G$ with $p, q, r \in W$, put the following transitions into Δ: $((p, p, \emptyset), a, (q, q, \emptyset))$, $((p, p, \emptyset), a, (q, q, W))$, $((p, r, W), a, (q, r, W))$ in the case that $p \neq r$ and $((p, p, W), a, (q, Succ(p, W), W))$. The set F is the set of all (p, p, W) with $p \in W$ and $W \in G$ (in particular, $W \neq \emptyset$). Now it is shown that $(P, \Sigma, (s, s, \emptyset), \Delta, F)$ is a nondeterministic Büchi automaton recognising L.

Given a word recognised by the Büchi automaton, there is an accepting run which goes infinitely often through a node of the form (p, p, W) with $p \in W$. When it is in (p, p, W), the next node is of the form (q, p', W) with $p' = Succ(p, W)$ and $q \in W$, the second parameter will remain p' until the run reaches (p', p', W) from which it will transfer to a state of the form (q', p'', W) with $p'' = Succ(p', W)$. This argument shows that the run will actually go through all states of the form (q, q, W) with $q \in W$ infinitely often and that the first component of the states visited after (p, p, W) will always be a member of Q. Hence, if one takes the first components of the run, then almost all of its states are in W and all states occur in W infinitely often and it forms a run in the given Muller automaton. Hence the given ω-word is recognised by the Muller automaton as well.

On the other hand, if one has a run $q_0 q_1 \ldots$ accepting an ω-word in the given Muller automaton and if W is the set of states visited infinitely often and if n is the position in the run from which onwards only states in W are visited, then one can translate the run of the Muller automaton into a run of the Büchi automaton as follows: $(q_0, q_0, \emptyset)$, $(q_1, q_1, \emptyset)$, $\ldots$, $(q_n, q_n, \emptyset)$, (q_{n+1}, q_{n+1}, W). For $m > n$ and the m-th state being (q_m, r_m, W) then the r_{m+1} of the next state (q_{m+1}, r_{m+1}, W) is chosen such that $r_{m+1} = Succ(W, r_m)$ in the case $q_m = r_m$ and $r_{m+1} = r_m$ in the case $q_m \neq r_m$. It can be seen that all the transitions are transitions of the Büchi automaton. Furthermore, one can see that the sequence of the r_m is not eventually constant, as for each r_m there is a $k \geq m$ with $q_k = r_m$, $r_k = r_m$ and $r_{m+1} = Succ(r_m, W)$. Hence one can conclude that the states of the form (p, p, W) with $p \in W$ are infinitely often visited in the run and the Büchi automaton has also an accepting run for the given word. Again the construction gives an exponential upper bound on the number of states of the Büchi automaton constructed from the Muller automaton. This completes the proof. ∎

In the above proof, an exponential upper bound on the number of states means that there is a polynomial f such that the number of states in the new automaton is bounded by $2^{f(|Q|)}$ where Q is the set of states of the old automaton and $|Q|$ denotes the number of states. So the algorithm gives the implicit bound that if one computes from some nondeterministic Büchi automaton another nondeterministic Büchi automaton recognising the complement then the number of states is going up in an double-exponential way, that is, there is some polynomial g such that the number of states in the Büchi automaton recognising the complement is bounded by $2^{2^{g(|Q|)}}$. This bound is not optimal, as the next result shows, but it can be improved to an exponential upper bound. Schewe [80] provides a tight exponential bound, the following theorem just takes the previous construction to give some (non-optimal) way to complement a Büchi automaton which still satisfies an exponential bound.

Theorem 6.20. *Assume that $(Q, \Sigma, \delta, s, F)$ is a nondeterministic Büchi automaton recognising the language L. Then there is an only exponentially larger automaton recognising the complement of L.*

Proof. For the given automaton for a language L, take the construction from Theorem 6.19 (a)⇒(b) to find a deterministic Muller automaton for L with a set P of states, a transition function Δ, $\tilde{p}$ and the numbers k and sets U_k defined as there. Now define the new state-space as $R = P \cup \{\tilde{p} \cup \{\hat{h}\} : \exists k\, [h \leq k < \infty \wedge k \in \tilde{p}]\}$ where $\hat{0}, \hat{1}, \ldots$ are considered as different from $0, 1, \ldots$ (for avoiding multi-sets) and the mapping $h \mapsto \hat{h}$ is one-one. So besides the states in P, there are states with an additional number $\hat{h}$ which is considered as a commitment that the value of the numbers in future states will never be below h. Note that $|R| \leq (2 * |Q| + 1) * |P|$ so that the exponential bound on R is only slightly larger than the one on P. The idea is that the transition relation on R follows in general Δ with the additional constraints, that at any time a commitment can be made but it can never be revised; furthermore, the commitment cannot be violated. So the following transitions are possible:

1. $(\tilde{p}, a, \Delta(\tilde{p}, a))$ if $\tilde{p} \in P$;
2. $(\tilde{p}, a, \Delta(\tilde{p}, a) \cup \{\hat{h}\})$ if $\tilde{p} \in P$ and h is any number between 0 and $2 * |Q|$;
3. $(\tilde{p} \cup \{\hat{h}\}, a, \Delta(\tilde{p}, a) \cup \{\hat{h}\})$ if $\tilde{p} \in P$ and there is a $h' \geq h$ with $h' \in \Delta(\tilde{p})$.

The set F has the role to enforce that for the limit inferior k of the numbers occurring in the states $\tilde{p}$, the commitment $\hat{k}$ is made eventually and some state $\tilde{p} \cup \{\hat{k}\}$ is visited infinitely often with $k \in \tilde{p} \wedge \tilde{p} \notin U_k$. Note that Theorem 6.19 (a)⇒(b) showed that this is exactly the behaviour of the underlying Muller automaton (without the commitments) when running on an ω-word: it rejects this ω-word iff it runs through a state $\tilde{p}$ infinitely often such that $k \in \tilde{p}$ for the limit inferior k of the numbers encoded into each of the infinitely often visited states and $\tilde{p} \notin U_k$. Indeed, all the states visited infinitely often are either all in U_k or all outside U_k. Therefore, one can choose F as follows:

$$F = \{\tilde{p} \cup \{\hat{k}\} : \tilde{p} \in P \wedge k \in \tilde{p} \wedge \tilde{p} \notin U_k\}.$$

Now the nondeterministic part of this Büchi automaton is to eventually guess k and do the corresponding commitment; if it then goes infinitely often through a node $\tilde{p} \cup \{\hat{k}\}$ of F, then the fact that $k \in \tilde{p}$ enforces that the limit inferior of the positions deleted in the strings is k; furthermore, the commitment enforces that it is not below k. Hence the simulated Muller automaton has the parameter k for its limit and cycles infinitely often through a node not in U_k; this implies that it rejects the ω-word and that the word is not in L. It is however accepted by the new Büchi automaton.

On the other hand, if the new Büchi automaton has only rejecting runs when chosing the right parameter k, then therefore all the states through which the Büchi automaton cycles through an infinite run with the right commitment are of the form $\tilde{p} \cup \{\tilde{k}\}$ with $\tilde{p} \in U_k$; hence the underlying Muller automaton accepts the ω-word and the Büchi automaton correctly rejects the ω-word. Hence the new Büchi automaton recognises the complement of L. ▮

Exercise 6.21. *Let $\Sigma = \{0, 1, 2\}$ and a parameter h be given. Make a nondeterministic Büchi automaton recognising the language L of all ω-words $b_1 b_2 \ldots$ in which there are infinitely many m such that $b_m = b_{m+h}$. Give a bound on the number of states of this automaton. Construct a Muller automaton recognising the same language and a Büchi automaton recognising the complement of L. How many states do these automata have (in terms of the value h)?*

Exercise 6.22. *Let $h = |\Sigma|$ and let L be any language where for each ω-word α the membership of α in L only depends on the set of symbols which appears infinitely often in α. Show that there is a deterministic Muller automaton with h states recognising L.*

Exercise 6.23. *Let L contain all ω-words $\alpha \in \Sigma^\omega$ for which at least half of the symbols in Σ occurs infinitely often.*

1. *Make a deterministic Büchi automaton recognising L with up to $2^{|\Sigma|-1}$ states.*
2. *Make a nondeterministic Büchi automaton recognising L with up to $|\Sigma|^2/2 + 2$ states.*

Description 6.24: Rabin and Streett Automata. Rabin and Streett automata are automata of the form $(Q, \Sigma, \delta, s, \Omega)$ where Ω is a set of pairs (E, F) of subsets of Q and a run on an ω-word $b_0 b_1 b_2 \ldots$ is a sequence $q_0 q_1 q_2 \ldots$ with $q_0 = s$ and $(q_n, b_n, q_{n+1}) \in \delta$ for all n; such a run is accepting iff the set $U = \{p \in Q : \exists^\infty n\, [p = q_n]\}$ of infinitely often visited nodes satisfies

- in the case of Rabin automata that $U \cap E \neq \emptyset$ and $U \cap F = \emptyset$ for one pair $(E, F) \in \Omega$;
- in the case of Streett automata that $U \cap E \neq \emptyset$ or $U \cap F = \emptyset$ for all pairs $(E, F) \in \Omega$.

Given a deterministic Rabin automaton $(Q, \Sigma, \delta, s, \Omega)$, the Streett automaton

$$(Q, \Sigma, \delta, s, \{(F, E) : (E, F) \in \Omega\})$$

recognises the complement of the Rabin automaton.

Example 6.25. Assume that an automaton with states $Q = \{q_0, q_1, \ldots, q_9\}$ on seeing digit d goes into state q_d. Then the condition Ω consisting of all pairs $(Q - \{q_d\}, \{q_d\})$ produces an Rabin automaton which accepts iff some digit d appears only finitely often in a given ω-word.

Assume that an automaton with states $Q = \{s, q_0, q_1, \ldots, q_9\}$, start state s and transition function δ given by $\delta(s, d) = q_d$ and if $d = e$ then $\delta(q_d, e) = d$ else $\delta(q_d, e) = s$. Let $E = \{q_0, q_1, \ldots, q_9\}$ and $F = \{s\}$ and $\Omega = \{(E, F)\}$. This automaton is a deterministic Rabin automaton which accepts all ω-words where exactly one digit occurs infinitely often.

Furthermore, if one takes $\Omega = \{(\emptyset, \{s\})\}$ then one obtains a Streett automaton which accepts exactly the ω-words where exactly one digit occurs infinitely often, as the automaton cannot obtain that a state in $\emptyset$ comes up infinitely often and therefore has to avoid that the state s is visited infinitely often. This happens exactly when one digit comes infinitely often.

Quiz 6.26. *Give an algorithm to translate a Büchi automaton into a Streett automaton.*

Exercise 6.27. *Assume that, for $k = 1, 2$, an ω-language L_k is recognised by a Streett automaton $(Q_k, \Sigma, s_k, \delta_k, \Omega_k)$. Prove that then there is a Streett automaton recognising $L_1 \cap L_2$ with states $Q_1 \times Q_2$, start state (s_1, s_2), transition relation $\delta_1 \times \delta_2$ and an Ω containing $|\Omega_1| + |\Omega_2|$ pairs. Here $(\delta_1 \times \delta_2)((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$. Explain how Ω is constructed.*

Description 6.28. A alternating Büchi automaton is an adjustment of an alternating finite automaton to the procssing of infinite words. So the automaton has states Q, input alphabet Σ, transition rules δ, start state s and final states F. The alternating Büchi automaton processes an ω-word w in a way similar to an alternating automaton processing finite words. There are three cases of transition rules for current state q and next input symbol a:

- $(q, a) \to r$: Next state is r;
- $(q, a) \to r \vee p$: Anke picks r or p;
- $(q, a) \to r \wedge p$: Boris picks r or p.

Now the alternating Büchi automaton accepts an ω-word w iff Anke has a winning strategy to ensure that the game always goes infinitely often through states from F, independently of what moves Boris does.

Example 6.29. The following alternating Büchi automaton has states $\{p, q, r\}$, alphabet $\{0, 1\}$ and recognises the ω-language $(\{0\}^* \cdot \{1\})^\omega$, that is, all ω-words with infinitely many 1:

state	type	0	1
p	start, rejecting	$p \wedge q \wedge r$	$q \vee r$
q	accepting	$p \wedge q \wedge r$	$p \vee r$
r	accepting	$p \wedge q \wedge r$	$p \vee q$

The idea for the verification is if a 0 comes then Boris can enforce that the game goes into a rejecting state and if a 1 comes then Anke can enforce that the game goes to an accepting state. enforce to go to go to an accepting or a rejecting node. If the symbol is a 1 then Anke decides where to go; if the symbol is a 0 then Boris decides where to go. Thus Anke can force the game to go infinitely often into an accepting states iff the ω-word has infinitely many digits 1.

Example 6.30. Assume a Büchi Game (G, E, s, W) is given. Now one can construct a Büchi alternating automaton as follows: The set G is the state set, the alphabet is $\{\text{Anke}, \text{Boris}\}$. Assume that for a node q the outgoing edges to nodes $p_1, p_2, \ldots, p_k$. Then the Büchi alternating automaton has the following transitions:
$\delta(q, \text{Anke}) = p_1 \vee p_2 \vee \ldots \vee p_k$;
$\delta(q, \text{Boris}) = p_1 \wedge p_2 \wedge \ldots \wedge p_k$.
So the input tells which player selects the next move. The accepting states of the Büchi AFA are those in W.

Now Anke has a winning strategy for the game (G, E, s, W) iff the Büchi alternating automaton accepts $(\text{Anke}\,\text{Boris})^\omega$.

Exercise 6.31. *Given a Büchi game* (G, E, s, W), *construct a deterministic Büchi automaton which reads plays (sequences of nodes visited by alternating moves of Anke and Boris) and which accepts iff all moves in the play are possible and Anke wins the play.*

Exercise 6.32. *Given a Büchi game* (G, E, s, W), *a nondeterministic Büchi automaton is using the states* $G \cup \{r\}$ *with* $W \cup \{r\}$ *being accepting and the input is every second node of a play, so if the play is* $s\, q_1\, q_2\, q_3\, q_4\, \ldots$ *then Anke does the moves to* $q_1, q_3, q_5, \ldots$ *and Boris to* $q_2, q_4, q_6, \ldots$*; now the game is that Anke reads* q_{2k} *with* $q_0 = s$ *and if one cannot go from the current state* q_{2k-1} *to* q_{2k} *then Anke moves to* r *else Anke moves to a successor node of* q_{2k} *which is called* q_{2k+1}. *From* r, *one can only move to* r *(independently of the input).*

Show that Anke has a winning strategy for the Büchi game iff the so constructed nondeterministic finite automaton accepts all halfplays as described here.

Exercise 6.33. *Construct the Büchi automaton from the previous exercise explicitly for the Büchi game given by* $G = \{1, 2, 3, 4\}$, $s = 1$, $W = \{2, 3\}$ *and* $E = \{(1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2, 1), (3, 3), (3, 1), (4, 4)\}$.

Selftest 6.34. *Assume that a finite game has the nodes* $\{00, 01, \ldots, 99\}$ *and the game can go from* ab *to* bc *for all* $a, b, c \in \{0, 1, \ldots, 9\}$*; except for the node* 00 *which is the target node. The player reaching* 00 *first wins the game. Which nodes are winning nodes for Anke and which are winning nodes for Boris and which are draw nodes?*

Selftest 6.35. *Consider a game on* $\mathbb{N}^3$ *where each player can move from* (a, b, c) *to* (a', b', c') *if either* $a = a' \wedge b = b' \wedge c = c' + 1$ *or* $a = a' \wedge b = b' + 1 \wedge c = 0$ *or* $a = a' + 1 \wedge b = 0 \wedge c = 0$*. The player which moves into* $(0, 0, 0)$ *loses the game, so other than in the previous task, each player has to try to avoid moving to* $(0, 0, 0)$*. Are there draw nodes in the game? Is every play of the game finite? The node* $(0, 0, 0)$ *is not counted.*

Selftest 6.36. *Let* $V = \{0, 1, \ldots, 19\}$ *and* $E = \{(p, q) : q \in \{p + 1, p + 2, p + 3\}$ *(modulo* 20*)*$\}$*; the starting node is* 0*. Does Anke have a memoryless winning strategy for the set* $\{6, 16\}$ *of nodes which must be visited infinitely often? If so, list out this memoryless winning strategy, if not, say why it does not exist.*

Selftest 6.37. *Explain the differences between a Büchi game, a survival game and an update game.*

Selftest 6.38. *Construct a nondeterministic Büchi automaton recognising the* ω*-language* $\{0, 1, 2\}^* \cdot \{001, 002\}^\omega \cup \{0, 1, 2\}^* \cdot \{01, 02\}^\omega$*.*

Selftest 6.39. *Construct a deterministic Büchi automaton which recognises the* ω*-language* $\{00\}^* \cdot \{11, 22\}^+ \cdot \{00\} \cdot \{22, 00\}^\omega$*.*

Selftest 6.40. *Prove that if a Rabin automaton recognises an* ω*-language* L *so does a Büchi automaton.*

Solution for Selftest 6.34. *The winning nodes for Anke are* $\{10, 20, \ldots, 90\}$ *where she moves to* 00*; for all other nodes, Anke must avoid that Boris can move to* 00 *and would move from* ab *to any number* bc *with* $c \neq 0$. *Boris does the same, hence all numbers of the form* ab *with* $b \neq 0$ *are draw nodes.*

Solution for Selftest 6.35. *One can see that every move of the game from* (a, b, c) *to* (a', b', c') *satisfies* $(a', b', c') <_{lex} (a, b, c)$ *for the lexicographic order on triples of numbers. As this ordering is, on* $\mathbb{N}^3$*, a well ordering, every play of the game must eventually go to* $(0, 0, 0)$ *and one of the players loses and the other one wins. So there are no infinite plays and thus also no draw nodes from which both players could enforce an infinite play.*

Solution for Selftest 6.36. *Yes, Anke has a memoryless winning-strategy and it is given by the following table:* $0 \to 3$, $1 \to 3$, $2 \to 3$, $3 \to 6$, $4 \to 6$, $5 \to 6$, $6 \to 7$, $7 \to 8$, $8 \to 9$, $9 \to 10$, $10 \to 13$, $11 \to 13$, $12 \to 13$, $13 \to 16$, $14 \to 16$, $15 \to 16$, $16 \to 17$, $17 \to 18$, $18 \to 19$, $19 \to 0$. *When the game is in node* p *with* $0 \leq p \leq 2$ *then it goes to Anke either in* p *or in some node* q *with* $p+1 \leq q \leq p+3$. *If it goes to Anke in node* p *then she moves to* 3*; now Boris can either move to* 6 *and satisfy the visit-requirement or to* 4 *or* 5 *in which case Anke satisfies the visit-requirement for* 6 *in the next move. If it is Boris move while the game is in* 0 *or* 1 *or* 2*, he can move it to* 1*,* 2*,* 3*,* 4 *or* 5 *and either Anke will move to* 3 *or to* 6 *directly, if the game goes to* 3 *it will also eventually visit* 6. *Similarly one shows when the game comes across some of the nodes* $10, 11, 12$ *it will eventually go to* 16 *with Anke's winning strategy. Thus Anke can play a memoryless winning strategy which enforces infinitely many visits of* 6 *and* 16 *in this update game.*

Solution for Selftest 6.37. *A survival game is a game where Anke wins if the game runs forever. A Büchi game has in addition to the set* V *of nodes also a subset* W *such that Anke wins iff the game runs forever and at least one node of* W *is visited infinitely often. Every survival game is also a Büchi game (by chosing* $W = V$*) but not vice versa. An update game is a game which has like a Büchi game a selected set* W *of nodes; the difference to the Büchi game is, however, that every node in* W *must be visited infinitely often. While Büchi games have always memoryless winning strategies, update games might fail to do so (at least for player Anke).*

Solution for Selftest 6.38. *The Büchi automaton has states* $q_0, q_1, q_2, q_3, q_4, q_5$ *and* q_0 *is the start state,* q_1, q_3 *are the accepting states and one can go on* $0, 1, 2$ *from* q_0 *to* q_0, q_1, q_3*, from* q_1 *on* 0 *to* q_2 *and from* q_2 *on* $1, 2$ *to* q_1*, from* q_3 *on* 0 *to* q_4*, from* q_4 *on* 0 *to* q_5 *and from* q_5 *on* $1, 2$ *to* q_3. *Thus when going to* q_1 *the automaton only will process further inputs from* $\{01, 02\}^\omega$ *and when going to* q_3*, the automaton will only process further inputs from* $\{001, 002\}^\omega$.

Solution for Selftest 6.39. *The table of the Büchi automaton looks as follows:*

state	*type*	0	1	2
q_0	*start,reject*	$q_{0,0}$	$q_{0,1}$	$q_{1,2}$
$q_{0,0}$	*reject*	q_0	–	–
$q_{0,1}$	*reject*	–	q_1	–
q_1	*reject*	$q_{2,0}$	$q_{1,1}$	$q_{1,2}$
$q_{1,1}$	*reject*	–	q_1	–
$q_{1,2}$	*reject*	–	–	q_1
q_2	*accept*	$q_{2,0}$	–	$q_{2,2}$
$q_{2,0}$	*reject*	q_2	–	–
$q_{2,2}$	*reject*	–	–	q_2

It accepts an infinite word iff it goes infinitely often through q_2. After the first time it went thorough this node, it will only process concatenations of 00 *and* 22*, as required.*

Solution for Selftest 6.40. *The idea is that for every e, E, F with $(E, F) \in \Omega$ and $e \in E$, one considers the regular language A_e of all words w such that the given Rabin automaton can go on w from the start state to e and the language $B_{e,F}$ of all non-empty words v such that the Rabin automaton can go from e to e on v without visiting any state in F. Now the automaton can on every ω-word from $A_e \cdot B_{e,F}^\omega$ go in the A_e-part of the ω-word from the start state to e and then in the $B_{e,F}^\omega$-part of the ω-word cycle from e to e without visiting any state from F. Thus there is an infinite run on the ω-word where the state e from E is visited infinitely often while no state from F is visited infinitely often and so $A_e \cdot B_{e,F}$ is a subset of the language recognised by the Rabin automaton. One can see that the language of all ω-words recognised by the Rabin automaton is the union of all $A_e \cdot B_{e,F}^\omega$ for which there is an E with $e \in E$ and $(E, F) \in \Omega$. So the ω-language recognised by the Rabin automaton is of the form from Theorem 6.14 and therefore recognised by a Büchi automaton.*

7 Automatic Functions and Relations

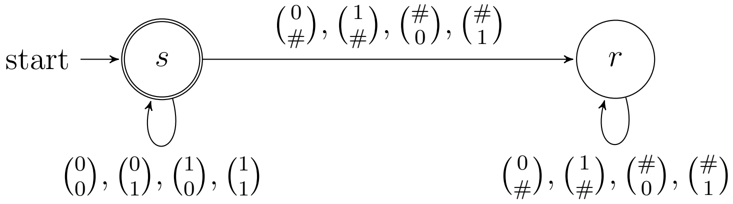
So far, only regular sets were considered. The notion of the regular sets has been generalised to automatic relations and functions.

Definition 7.1: Automatic Relations and Functions [5, 41, 42, 53]**.** *A relation $R \subseteq X \times Y$ is automatic iff there is an automaton reading both inputs at the same speed (one symbol per cycle with a special symbol # given for inputs which are exhausted) such that $(x, y) \in R$ iff the automaton is in an accepting state after having read both, x and y, completely.*

Similarly one can define that a relation of several parameters is automatic.

A function $f : X \to Y$ is automatic iff the relation $\{(x, y) : x \in dom(f) \wedge y = f(x)\}$ is automatic.

Example 7.2. The relation $|x| = |y|$ is an automatic relation, given by an automaton which remains in an accepting starting state as long as both inputs are in Σ and transfers to a rejecting state (which it does not leave) whenever exactly one of the inputs x, y is exhausted.


start
s
r
$\binom{0}{\#}, \binom{1}{\#}, \binom{\#}{0}, \binom{\#}{1}$
$\binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{1}{1}$
$\binom{0}{\#}, \binom{1}{\#}, \binom{\#}{0}, \binom{\#}{1}$

Here $\binom{a}{b}$ means that the first input (x) provides an a and the second input (y) provides a b; the alphabet is $\{0, 1\}$.

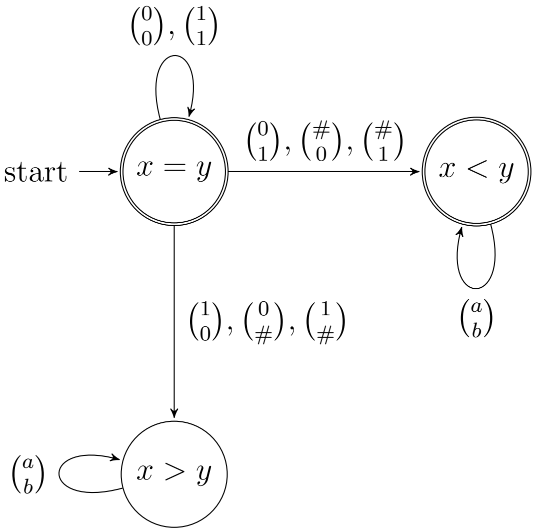
For example, if $x = 00$ and $y = 1111$, then the automaton starts in s, reads $\binom{0}{1}$ and remains in s, reads $\binom{0}{1}$ and remains in s, reads $\binom{\#}{1}$ and goes to r, reads $\binom{\#}{1}$ and remains in state r. As now both inputs are exhausted and the automaton is in a rejecting state, it rejects the input; indeed, $|00| \neq |1111|$.

If $x = 010$ and $y = 101$, then the automaton starts in s and remains in s while processing $\binom{0}{1}$, $\binom{1}{0}$ and $\binom{0}{1}$.

Notation 7.3. A set of pairs, or in general of tuples, can be directly be written in the way as the automaton would read the inputs. For this let $conv(x, y)$ be the set of pairs with symbols from x and y at the same position, where # is used to fill the shorter string (if applicable). So $conv(00, 111) = \binom{0}{1}\binom{0}{1}\binom{\#}{1}$, $conv(101, 222) = \binom{1}{2}\binom{0}{2}\binom{1}{2}$ and $conv(0123, 33) = \binom{0}{3}\binom{1}{3}\binom{2}{\#}\binom{3}{\#}$. For this, it is always understood that the symbol # is not in the alphabet used for x and y; if it would be, some other symbol has to be used for denoting the empty places of the shorter word. So a relation $R \subseteq \Sigma^* \times \Sigma^*$ is automatic iff the set $\{conv(x,y) : (x,y) \in R\}$ is regular. Similarly for relations with an arity other than two.

Example 7.4. Assume that the members of Σ are ordered; if $\Sigma = \{0,1\}$ then the default ordering is $0 < 1$. One says that a string $v = a_1 a_2 \ldots a_n$ is lexicographic before a string $w = b_1 b_2 \ldots b_m$ iff either $n < m$ and $a_1 = b_1 \wedge a_2 = b_2 \wedge \ldots \wedge a_n = b_n$ or there is a $k < \min\{n,m\}$ with $a_1 = b_1 \wedge a_2 = b_2 \wedge \ldots \wedge a_k = b_k \wedge a_{k+1} < b_{k+1}$. One writes $v <_{lex} w$ if v is lexicographic before w; $v \leq_{lex} w$ means either $v <_{lex} w$ or $v = w$. So $000 <_{lex} 00110011 <_{lex} 0101 <_{lex} 010101 <_{lex} 1 <_{lex} 10 <_{lex} 100 <_{lex} 11$.

The lexicographic ordering is an automatic relation. For the binary alphabet $\{0,1\}$, it is recognised by the following automaton.


$\binom{0}{0}, \binom{1}{1}$
start
$x = y$
$\binom{0}{1}, \binom{\#}{0}, \binom{\#}{1}$
$x < y$
$\binom{a}{b}$
$\binom{1}{0}, \binom{0}{\#}, \binom{1}{\#}$
$\binom{a}{b}$
$x > y$

Here $\binom{a}{b}$ on an arrow means that the automaton always goes this way.

Exercise 7.5. *Say in words which automatic relations are described by the following regular expressions:*

- $\{\binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{1}{1}\}^* \cdot \{\binom{0}{0}, \binom{1}{1}\} \cdot (\{\binom{\#}{0}, \binom{\#}{1}\}^* \cup \{\binom{0}{\#}, \binom{1}{\#}\}^*)$,
- $\{\binom{0}{1}, \binom{1}{0}\}^* \cdot (\{\varepsilon\} \cup \{\binom{2}{\#}\} \cdot \{\binom{0}{\#}, \binom{1}{\#}, \binom{2}{\#}\}^*)$,
- $\{\binom{0}{0}, \binom{1}{0}, \binom{2}{0}, \ldots, \binom{9}{0}\}^*$*?*

Which of these three relations define functions? What are the domains and ranges of these functions?

Exercise 7.6. *Which of the following relations are automatic (where x_k is the k-th symbol of $x = x_1 x_2 \ldots x_n$ and $|x| = n$):*

- $R_1(x,y,z) \Leftrightarrow \forall k \in \{1,2,\ldots,\min\{|x|,|y|,|z|\}\}\,[x_k = y_k \vee x_k = z_k \vee y_k = z_k]$;
- $R_2(x,y,z) \Leftrightarrow |x| + |y| = |z|$;
- $R_3(x,z) \Leftrightarrow \exists y\,[|x| + |y| = |z|]$;
- $R_4(x,y,z) \Leftrightarrow \exists k \in \{1,2,\ldots,\min\{|x|,|y|,|z|\}\}\,[x_k = y_k = z_k]$;
- $R_5(x,y,z) \Leftrightarrow \exists i,j,k\,[x_i = y_j = z_k]$;
- $R_6(x,y) \Leftrightarrow y = 012 \cdot x \cdot 012$.

Give a short explanations why certain relations are automatic or not; it is not needed to construct the corresponding automata by explicit tables or diagrams.

Theorem 7.7: First-Order Definable Relations [53]. *If a relation is first-order-definable using automatic functions and relations then it is automatic; if a function is first-order-definable using automatic functions and relations, then it is automatic. Furthermore, one can construct the automata effectively from the automata used in the parameters to define the relation or function.*

Example 7.8. The length-lexicographic or military ordering can be defined from the two previously defined orderings: $v <_{ll} w$ iff $|v| < |w|$ or $|v| = |w| \wedge v <_{lex} w$. Hence the length-lexicographic ordering is automatic.

Furthermore, for every automatic function f and any regular subset R of the domain of f, the image $f(R) = \{f(x) : x \in R\}$ is a regular set as well, as it is first-order definable using f and R as parameters:

$$y \in f(R) \Leftrightarrow \exists x \in R\,[f(x) = y].$$

Exercise 7.9. *Let I be a regular set and $\{L_e : e \in I\}$ be an automatic family, that is, a family of subsets of Σ^* such that the relation of all (e,x) with $e \in I \wedge x \in L_e$ is automatic. Note that $D = \bigcup_{i \in I} L_i$ is first-order definable by*

$$x \in D \Leftrightarrow \exists i \in I\,[x \in L_i],$$

hence D is a regular set. Show that the following relations between indices are also automatic:

- $\{(i,j) \in I \times I : L_i = L_j\}$;
- $\{(i,j) \in I \times I : L_i \subseteq L_j\}$;
- $\{(i,j) \in I \times I : L_i \cap L_j = \emptyset\}$;
- $\{(i,j) \in I \times I : L_i \cap L_j \text{ is infinite}\}$.

Show this by showing that the corresponding relations are first-order definable from given automatic relations. One can use for the fourth the length-lexicographic order in the first-order definition.

Example 7.10. Let (N, Σ, P, S) be a grammar and $R = \{(x, y) \in (N \cup \Sigma)^* \times (N \cup \Sigma)^* : x \Rightarrow y\}$ be the set of all pairs of words where y can be derived from x in one step. The relation R is automatic.

Furthermore, for each fixed n, the relation $\{(x, y) : \exists z_0, z_1, \ldots, z_n\, [x = z_0 \wedge y = z_n \wedge z_0 \Rightarrow z_1 \wedge z_1 \Rightarrow z_2 \wedge \ldots \wedge z_{n-1} \Rightarrow z_n]\}$ of all pairs of words such that y can be derived from x in exactly n steps is automatic.

Similarly, the relation of all (x, y) such that y can be derived from x in at most n steps is automatic.

Remark 7.11. In general, the relation $\Rightarrow^*$ is not automatic for a non-regular grammar, even if the language generated by the grammar itself is regular. For example, one could consider the grammar $(\{S\}, \{0, 1, 2\}, \{S \to SS|0|1|2\}, S)$ generating all non-empty words over $\{0, 1, 2\}$. Then consider a derivation $S \Rightarrow^* S01^m2S \Rightarrow^* 0^k1^m2^n$. If $\Rightarrow^*$ would be automatic, so would be the relation of all pairs of the form $(S01^m2S, 0^k1^m2^n)$ with $k > 1 \wedge m > 0 \wedge n > 1$; this is the set of those pairs in $\Rightarrow^*$ where the first component is of the form $S011^*2S$. If the set of the convoluted pairs in $\Rightarrow^*$ is regular, so is this set.

Now, choose $n = h + 4$, $m = h$, $k = h + 4$ for a h much larger than the pumping constant of the assumed regular set; then the regular set of the convoluted pairs in the relation would contain for every r the string

$$\binom{S}{0}\binom{0}{0}\binom{1}{0}^{c}\binom{1}{0}^{dr}\binom{1}{0}^{h-c-d}\binom{2}{0}\binom{S}{0}\binom{\#}{1}^{h}\binom{\#}{2}^{h+4};$$

where $c \geq 0$, $d > 0$ and $h - c - d \geq 0$. In contrast to this, the condition on $\Rightarrow^*$ implies that the first S is transformed in a sequence of 0 and the second S into a sequence of 2 while the number of 1 is preserved; therefore the number $c + dr + h - c - d$ must be equal to h, which gives a contradiction for $r \neq 1$. Hence the relation $\Rightarrow^*$ cannot be automatic.

It should however be noted, that the relation $\Rightarrow^*$ is regular in the case that the grammar used is regular. In this case, for N denoting the non-terminal and Σ the terminal alphabet and for every $A, B \in N$, let $L_{A,B}$ denote the set of all words w such that $A \Rightarrow^* wB$ and L_A be the set of all words w such that $A \Rightarrow^* w$. All sets L_A and $L_{A,B}$ are regular and now $x \Rightarrow^* y$ iff either $x = y$ or $x = vA$ and $y = vwB$ with $w \in L_{A,B}$ or $x = vA$ and $y = vw$ with $w \in L_A$ for some $A, B \in N$; hence the convoluted pairs of the relation $\Rightarrow^*$ form the union

$$\{conv(x,y) : x \Rightarrow^* y\} = \bigcup_{A,B \in N} (\{conv(vA, vwB) : v \in \Sigma^*, w \in L_{A,B}\} \;\cup \\ \{conv(vA, vw) : v \in \Sigma^*, w \in L_A\})$$

which is a regular set.

Exercise 7.12. *Let R be an automatic relation over $\Sigma^* \cup \Gamma^*$ such that whenever $(v,w) \in R$ then $|v| \leq |w|$ and let L be the set of all words $x \in \Sigma^*$ for which there exists a sequence $y_0, y_1, \ldots, y_m \in \Gamma^*$ with $y_0 = \varepsilon$, $(y_k, y_{k+1}) \in R$ for all $k < m$ and $(y_m, x) \in R$. Note that $\varepsilon \in L$ iff $(\varepsilon, \varepsilon) \in R$. Show that L is context-sensitive.*

Note that the converse direction of the statement in the exercise is also true. So assume that L is context-sensitive. Then one can take a grammar for $L - \{\varepsilon\}$ where each rule $v \to w$ satisfies $|v| \leq |w|$ and either $v, w \in N^+$ or $|v| = |w| \wedge v \in N^+ \wedge w \in \Sigma^+$. Now let $(x,y) \in R$ if either $x, y \in N^+ \wedge x \Rightarrow y$ or $x \in N^+ \wedge y \in \Sigma^+ \wedge (x \Rightarrow^* y$ by rules making non-terminals to terminals) or $(x,y) = (\varepsilon,\varepsilon) \wedge \varepsilon \in L$.

Theorem 7.13: Immerman and Szelepcsényi's Nondeterministic Counting [44, 84]. *The complement of a context-sensitive language is context-sensitive.*

Proof. The basic idea is the following: Given a word x of length n and a context-sensitive grammar (N, Σ, P, S) generating the language L, there is either a derivation of x without repetition or there is no derivation at all. One can use words over $\Sigma \cup N$ to represent the counter values for measuring the length of the derivation as well as the number of counted values; let u be the largest possible counter value. Now one determines using nondeterminism for each ℓ how many words can be derived with derivations up to length ℓ; the main idea is that one can obtain the number for $\ell + 1$ from the number for ℓ and that the number for $\ell = 0$ is 1 (namely the start symbol S). The idea is to implement a basic algorithm is the following:

```
Choose an $x \in \Sigma^*$ and verify that $x \notin L$ as follows;
Let $u$ be the length-lexicographically largest string in $(N \cup \Sigma)^{|x|}$;
Let $i = Succ_{ll}(\varepsilon)$;
For $\ell = \varepsilon$ to $u$ Do Begin

    Let $j = \varepsilon$;
    For all $y \leq_{ll} u$ Do Begin
        Derive the words $w_1, w_2, \ldots, w_i$ nondeterminis-
            tically in length-lexicographic order in up to
            $\ell$ steps each and do the following checks:
```

```
        If there is a $w_m$ with $w_m \Rightarrow y$ or $w_m = y$ then
            let $j = Succ_{ll}(j)$;
        If there is a $w_m$ with $w_m = x$ or $w_m \Rightarrow x$ then
            abort the computation End;
    Let $i = j$; End;

If the algorithm has not yet aborted then generate $x$;
```

This algorithm can be made more specific by carrying over the parts which use plain variables and replacing the indexed variables and meta-steps. For this, the new variables v, w to run over the words and k to count the derivation length; h is used to count the words processed so far. Recall that the special case of generating or not generating the empty word is ignored, as the corresponding entry can be patched easily in the resulting relation R, into which the algorithm will be translated.

```
1: Choose an $x \in \Sigma^+$ and initial all other variables as $\varepsilon$;
2: Let $u = (\max_{ll}(N \cup \Sigma))^{|x|}$;
3: Let $i = Succ_{ll}(\varepsilon)$ and $\ell = \varepsilon$;
4: While $\ell <_{ll} u$ Do Begin
    5: Let $j = \varepsilon$;
    6: Let $y = \varepsilon$;
    7: While $y <_{ll} u$ Do Begin
        8: Let $y = Succ_{ll}(y)$;
        9: Let $h = \varepsilon$ and $w = \varepsilon$;
        10: While $h <_{ll} i$ Do Begin
            11: Nondeterministically replace $w$ by $w'$ with $w <_{ll} w' \leq_{ll} u$;
            12: Let $v = S$;
            13: Let $k = \varepsilon$;
            14: While $(v \neq w) \wedge (k <_{ll} \ell)$ Do Begin
                15: Nondeterministically replace $(k, v)$ by $(k', v')$ with $k <_{ll} k'$ and
                    $v \Rightarrow v'$ End;
            16: If $v \neq w$ Then abort the computation (as it is spoiled);
            17: If $w = x$ or $w \Rightarrow x$ Then abort the computation (as $x \in L$);
            18: If $w \neq y$ and $w \not\Rightarrow y$
                19: Then let $h = Succ_{ll}(h)$;
                **20:** Else let $h = i$;
            **21:** End (of While Loop in 10);
        **22:** If $w = y$ or $w \Rightarrow y$ Then $j = \mathit{Succ}_{ll}(j)$;
        **23:** End (of While Loop in 7);
    **24:** Let $i = j$;
    **25:** Let $\ell = \mathit{Succ}_{ll}(\ell)$ End (of While Loop in 4);
**26:** If the algorithm has not yet aborted Then generate $x$;
```

The line numbers in this algorithm are for reference and will be used when making the relation R. Note that the algorithm is nondeterministic and that x is generated iff some nondeterministic path through the algorithm generates it. Pathes which lose their control information are just aborted so that they do not generate false data.

The variables used in the algorithm are $h, i, j, k, \ell, u, v, w, x, y$ whose values range over $(N \cup \Sigma)^*$ and furthermore, one uses a for the line numbers of the algorithm where a takes one-symbol words representing line numbers from the set $\{4, 7, 10, 14, 16, 18, 22, 24, 26\}$. Now the relation R is binary and connects words in Σ^* with intermediate non-terminal words represented by convolutions of the form $\mathit{conv}(a, h, i, j, k, \ell, u, v, w, x, y)$. The relation contains all the pairs explicitly put into R according to the following list, provided that the "where-condition" is satisfied. The arrow "$\rightarrow$" indicates the order of the two components of each pair.

Input: $\varepsilon \rightarrow \mathit{conv}(4, h, i, j, k, \ell, u, v, w, x, y)$ where $h = \varepsilon$, $i = \mathit{Succ}_{ll}(\varepsilon)$, $j = \varepsilon$, $k = \varepsilon$, $\ell = \varepsilon$, $u \in (\max(N \cup \Sigma))^+$, $v = \varepsilon$, $w = \varepsilon$, $x \in \Sigma^{|u|}$, $y = \varepsilon$;

4: Put $\mathit{conv}(4, h, i, j, k, \ell, u, v, w, x, y) \rightarrow \mathit{conv}(26, h, i, j, k, \ell, u, v, w, x, y)$ into R where $\ell = u$ (for the case where the while-loop terminates);
Put $\mathit{conv}(4, h, i, j, k, \ell, u, v, w, x, y) \rightarrow \mathit{conv}(7, h, i, \varepsilon, k, \mathit{Succ}_{ll}(\ell), u, v, w, x, \varepsilon)$ into R where $\ell <_{ll} u$ (for the case there another round of the loop is started with resetting j and y in lines 5 and 6 and then continuing in line 7);

7: Put $\mathit{conv}(7, h, i, j, k, \ell, u, v, w, x, y) \rightarrow \mathit{conv}(24, h, i, j, k, \ell, u, v, w, x, y)$ into R where $y = u$ (for the case where the while-loop terminates);
Put $\mathit{conv}(7, h, i, j, k, \ell, u, v, w, x, y) \rightarrow \mathit{conv}(10, \varepsilon, i, j, k, \ell, u, v, \varepsilon, x, \mathit{Succ}_{ll}(y))$ into R (for the case that the while-loop starts and y, h, w are updated in lines 8,9);

10: Put $\mathit{conv}(10, h, i, j, k, \ell, u, v, w, x, y) \rightarrow \mathit{conv}(22, h, i, j, k, \ell, u, v, w, x, y)$ into R where $h = i$ (for the case where the while-loop terminates);
Put $\mathit{conv}(10, h, i, j, k, \ell, u, v, w, x, y) \rightarrow \mathit{conv}(14, h, i, j, \varepsilon, \ell, u, S, w', x, y)$ into R

where $h <_{ll} i$ and $w <_{ll} w' \leq_{ll} u$ (for the case that the body of the loop is started and the commands in lines 11, 12 and 13 are done);

14: Put $conv(14, h, i, j, k, \ell, u, v, w, x, y) \rightarrow conv(14, h, i, j, k', \ell, u, v', w, x, y)$ into R where $v \neq w$ and $k <_{ll} k' \leq_{ll} \ell$ and $v \Rightarrow v'$ (for the case that the body of the loop in line 15 is done one round);
Put $conv(14, h, i, j, k, \ell, u, v, w, x, y) \rightarrow conv(18, h, i, j, k, \ell, u, v, w, x, y)$ into R where $v = w$ and $w \neq x$ and $w \not\Rightarrow x$ (for the case that the loop leaves to line 18 without the computation being aborted);

18: Put $conv(18, h, i, j, k, \ell, u, v, w, x, y) \rightarrow conv(10, Succ_{ll}(h), i, j, k, \ell, u, v, w, x, y)$ into R where $w \neq y$ or $w \not\Rightarrow y$ (in the case that the program goes through the then-case);
Put $conv(18, h, i, j, k, \ell, u, v, w, x, y) \rightarrow conv(10, i, i, j, k, \ell, u, v, w, x, y)$ into R where $w = y$ or $w \Rightarrow y$ (in the case that the program goes through the else-case);

22: Put $conv(22, h, i, j, k, \ell, u, v, w, x, y) \rightarrow conv(7, h, i, Succ_{ll}(j), k, \ell, u, v, w, x, y)$ into R where $w = y$ or $w \Rightarrow y$ (for the case that the condition of line 22 applies before going to line 7 for the next round of the loop);
Put $conv(22, h, i, j, k, \ell, u, v, w, x, y) \rightarrow conv(7, h, i, j, k, \ell, u, v, w, x, y)$ into R where $w \neq y$ and $w \not\Rightarrow y$ (for the case that the condition of line 22 does not apply and the program goes to line 7 for the next round of the loop directly);

24: Put $(22, h, i, j, k, \ell, u, v, w, x, y) \rightarrow conv(4, h, j, j, k, Succ_{ll}(\ell), u, v, w, x, y)$ into R (which reflects the changes from lines 24 and 25 when completing the body of the loop starting in line 4);

Output: Put $(26, h, i, j, k, \ell, u, v, w, x, y) \rightarrow x$ into R (which produces the output after the algorithm has verified that $x \notin L$);

Special Case: Put $\varepsilon \rightarrow \varepsilon$ into R iff $\varepsilon \notin L$.

The verification that this relation is automatic as well as the proof of Exercise 7.12 are left to the reader. ∎

Exercise 7.14. *Consider the following algorithm to generate all non-empty strings which do not have as length a power of* 2.

```
1. Guess $x \in \Sigma^+$; Let $y = 0$;
2. If $|x| = |y|$ then abort;
3. Let $z = y$;
4. If $|x| = |y|$ then generate $x$ and halt;
```

```
5. Remove last 0 in z;
6. Let y = y0;
7. If z = ε then goto 2 else goto 4.
```

Make an R as in Exercise 7.12 choosing Γ such that Γ^* contains all strings in

$$\{conv(line, x, y, z) : line \in \{1, 2, 3, 4, 5, 6, 7\} \wedge x \in \Sigma^+ \wedge y, z \in \{0\}^*\}$$

but no non-empty string of Σ^* – the symbols produced by the convolution of these alphabets are assumed to be different from those in Σ. A pair (v, w) should be in R iff $|v| \leq |w|$ and one of the following conditions hold:

1. *$v = \varepsilon$ and w is a possible outcome of line 1, that is, of the form $conv(2, x, 0, \varepsilon)$ for some $x \in \Sigma^+$;*
2. *v is the configuration of the machine before executing the line coded in v and w the configuration after executing this line where the line number in w is either the next one after the one in v or a line number to which the program has branched by a condition or unconditional goto-command (whatever is applicable);*
3. *v is the configuration in line 4 and w is the value of x, provided that y and x have the same length and that x is as long as $conv(4, x, y, z)$.*

Give a full list of all the pairs which can occur such that, when starting from ε and iterating along the relation in R, exactly those $x \in \Sigma^*$ can be reached which do not have a length of a power of 2.

The following dfas are supposed to compute automatic functions, that is, to check whether the value y is the output of x under the corresponding function.

Exercise 7.15. *Let $x = a_0a_1 \ldots a_n$ be a ternary number representing $\sum_{m \leq n} 3^m \cdot a_m$ over the alphabet $\{0, 1, 2\}$. Construct a dfa which is correct on all convolutions $conv(x, y)$ and which checks whether $y = x + 1$.*

Exercise 7.16. *Let $x = a_0a_1 \ldots a_n$ be a ternary number representing $\sum_{m \leq n} 3^m \cdot a_m$ over the alphabet $\{0, 1, 2\}$. Construct a dfa which is correct on all convolutions $conv(x, y)$ and which checks whether $y = x + x + x + x$.*

Exercise 7.17. *Let $x = a_0a_1 \ldots a_n$ be a ternary number representing $\sum_{m \leq n} 3^m \cdot a_m$ over the alphabet $\{0, 1, 2\}$. Construct a dfa which is correct on all convolutions $conv(x, y)$ and which checks whether $y = x + x + 1$.*

Exercise 7.18. *Let $x = a_0a_1 \ldots a_n$ be a ternary number representing $\sum_{m \leq n} 3^m \cdot a_m$ over the alphabet $\{0, 1, 2\}$. Construct a dfa which is correct on all convolutions $conv(x, y)$ and which checks whether $y = 3^{n+1} - x - 1$.*

Exercise 7.19. *Let $x = a_0 a_1 \ldots a_n$ be a ternary number representing $\sum_{m \leq n} 3^m \cdot a_m$ over the alphabet $\{0, 1, 2\}$. Construct a dfa which is correct on all convolutions conv(x, y) and which checks whether $y = (x - a_0)/3 + 3^n \cdot a_0$.*

Exercise 7.20. *Let $x = a_0 a_1 \ldots a_n$ be a ternary number representing $\sum_{m \leq n} 3^m \cdot a_m$ over the alphabet $\{0, 1, 2\}$. Construct a dfa which is correct on all convolutions conv(x, y) and which checks whether $y = Even(x)$ where $Even(x)$ is 1 if x is even and is 0 if x is odd.*

8 Groups, Monoids and Automata Theory

There are quite numerous connections between group theory and automata theory. On one hand, concepts from group theory are used in automata theory; on the other hand, one can also use automata theory to describe certain types of groups and semigroups. First the basic definitions.

Definition 8.1: Groups and Semigroups. *Let G be a set and $\circ$ be an operation on G, that is, $\circ$ satisfies for all $x, y \in G$ that $x \circ y \in G$.*
(a) *The structure $(G, \circ)$ is called a semigroup iff $x \circ (y \circ z) = (x \circ y) \circ z$ for all $x, y, z \in G$, that is, if the operation $\circ$ on G is associative.*
(b) *The structure $(G, \circ, e)$ is called a monoid iff $e \in G$ and $(G, \circ)$ is a semigroup and e satisfies $x \circ e = e \circ x = x$ for all $x \in G$.*
(c) *The structure $(G, \circ, e)$ is called a group iff $(G, \circ, e)$ is a monoid and for each $x \in G$ there is an $y \in G$ with $x \circ y = e$.*
(d) *A semigroup $(G, \circ)$ is called finitely generated iff there is a finite subset $F \subseteq G$ such that for each $x \in G$ there are n and $y_1, y_2, \ldots, y_n \in F$ with $x = y_1 \circ y_2 \circ \ldots \circ y_n$.*
(e) *A semigroup $(G, \circ)$ is finite iff G is finite as a set.*

Example 8.2. The set $(\{1, 2, 3, \ldots\}, +)$ forms a semigroup, the neutral element 0 is not in the base set and therefore it is not a monoid. Adding 0 to the set and using $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ gives the additive monoid $(\mathbb{N}, +, 0)$ of the natural numbers. The integers $(\mathbb{Z}, +, 0)$ form a group; similarly the rationals with addition, denoted by $(\mathbb{Q}, +, 0)$. Also the multiplicative structure $(\mathbb{Q} - \{0\}, *, 1)$ of the non-zero rationals is a group.

Example 8.3. A semigroup with a neutral element only from one side does not necessarily have a neutral element from the other side, furthermore, the neutral element from one side does not need to be unique.

To see this, consider $G = \{0, 1\}$ and the operation $\circ$ given by $x \circ y = x$. This operation is associative, as $x \circ (y \circ z) = x \circ y = x$ and $(x \circ y) \circ z = x \circ z = x$. Both elements, $y = 0$ and $y = 1$, are neutral from the right side: $x \circ y = x$. On the other hand, none of these two elements is neutral from the left side, as the examples $0 \circ 1 = 0$ and $1 \circ 0 = 1$ show.

Example 8.4. Let Q be a finite set and G be the set of all functions from Q to Q. Let $f \circ g$ be the function obtained by $(f \circ g)(q) = g(f(q))$ for all $q \in Q$. Let id be the identity function ($id(q) = q$ for all $q \in Q$). Then $(G, \circ, id)$ is a finite monoid.

Let $G' = \{f \in G : f \text{ is one-one}\}$. Then $(G', \circ, id)$ is a group. The reason is that for every $f \in G'$ there is an inverse f^{-1} with $f^{-1}(f(q)) = q$ for all $q \in Q$. Indeed, $G' = \{f \in G : \exists g \in G\, [id = f \circ g]\}$.

Example 8.5. Let $(Q, \Sigma, \delta, s, F)$ be a complete dfa and G be the set of all mappings f from Q to Q and id be the identity function. Now one considers the set

$$G' = \{f \in G : \exists w \in \Sigma^* \, \forall q \in Q \, [\delta(q, w) = f(q)]\}$$

which is the monoid of those functions which are all realised as operations given by a word in Σ^*. Then $(G', \circ, id)$ is a monoid as well. Note that if f is realised by v and g is realised by w then $f \circ g$ is realised by $w \cdot v$. The identity id is realised by the empty word ε.

This monoid defines an equivalence-relation on the strings: Every function realises only one string, so one can chose for each string w the function $f_w \in G'$ realised by w. Now let $v \sim w$ iff $f_v = f_w$. This equivalence relation partitions Σ^* into finitely many equivalence classes (as G' is finite).

Furthermore, $v \sim w$ and $x \sim y$ imply $vx \sim wy$: Given any $q \in Q$, it is mapped by f_{vx} to $f_x(f_v(q))$ and by f_{wy} to $f_y(f_w(q))$. As $f_v = f_w$ and $f_x = f_y$, it follows that $f_{vx}(q) = f_{wy}(q)$. Hence $f_{vx} = f_{wy}$ and $vx \sim wy$. A relation $\sim$ with this property is called a congruence-relation on all strings.

Let L be the language recognised by the given dfa. Recall the definition $L_x = \{y : xy \in L\}$ from Theorem 2.19. Note that $x \sim y$ implies that $L_x = L_y$: Note that $f_x(s) = f_y(s)$ and therefore $\delta(s, xz) = \delta(s, yz)$ for all z; hence $z \in L_x$ iff $z \in L_y$ and $L_x = L_y$. Therefore L is the union of some equivalence classes of $\sim$.

If the dfa is the smallest-possible dfa, that is, the dfa produced in the proof of Theorem 2.19, then the monoid $(G', \circ, id)$ derived from it is called the syntactic monoid of the language L.

The syntactic monoid is a widely used tool in formal language theory. An immediate consequence of the ideas laid out in the above example is the following theorem.

Theorem 8.6. *A language is regular iff it is the finite union of equivalence classes of some congruence relation with finitely many congruence classes.*

Example 8.7. Let a dfa have the alphabet $\Sigma = \{0, 1\}$ and states $Q = \{s, s', q, q', q''\}$ and the following state transition table:

state	s	s'	q	q'	q''
successor at 0	s'	s	q	q'	q''
successor at 1	q'	q'	q'	q''	q

Now the syntactic monoid contains the transition functions f_ε (the identity), f_0, f_1, f_{11} and f_{111}. The functions f_0 and f_1 are as in the state transition diagramme and f_{11} and f_{111} are given by the following table.

state	s	s'	q	q'	q''
f_{11}	q''	q''	q''	q	q'
f_{111}	q	q	q	q'	q''

Other functions are equal to these, for example $f_{110} = f_{11}$ and $f_{0000} = f_\varepsilon$.

Definition 8.8. *Let $(G, \circ)$ be a semigroup and $w \in G^*$ be a string of elements in G. Then $el_G(w)$ denotes that element of G which is represented by w. Here $el_G(\varepsilon)$ denotes the neutral element (if it exists) and $el_G(a_1 a_2 \ldots a_n)$ the group element $a_1 \circ a_2 \circ \ldots \circ a_n$. Furthermore, if R is a set of strings over G^* such that for every $x \in G$ there is exactly one $w \in R$ with $el_G(w) = x$, then one also uses the notation $el_R(v) = w$ for every $v \in G^*$ with $el_G(v) = el_G(w)$.*

Let $F \subseteq G$ be a finite set of generators of G, that is, F is finite and satisfies for every $x \in G$ that there is a word $w \in F^*$ with $el_G(w) = x$. Now the word-problem of G asks for an algorithm which checks for two $v, w \in F^*$ whether $el_G(v) = el_G(w)$.

Definition 8.9. A language L defines the congruence $\{(v, w) : L_v = L_w\}$ and furthermore L also defines the syntactic monoid $(G_L, \circ)$ of its minimal deterministic finite automaton.

Theorem 8.10. *Every finite group $(G, \circ)$ is the syntactic monoid of a language L.*

Proof. Let $L = \{v \in G^* : el_G(v) = \varepsilon\}$ be the set of words which are equal to the neutral element.

The corresponding dfa is $(G, G, \delta, s, \{s\})$ with $\delta(a, b) = a \circ b$ for all $a, b \in G$; here the same names are used for the states and for the symbols. In order to avoid clashes with the empty word, the neutral element of G is called s for this proof; it is the start symbol and also the only accepting symbol.

For each $a \in G$, the inverse b of a satisfies that b is the unique one-letter word in G^* such that $L_{ab} = L$. Therefore, for all $a \in G$, the languages L_a are different, as two different elements $a, b \in G$ have different inverses. Thus the dfa is a minimal dfa and has a congruence $\sim$ satisfying $v \sim w$ iff $el_G(v) = el_G(w)$ iff $L_v = L_w$. Note that every word $w \in G^*$ satisfies $el_G(w) = a$ for some $a \in G$. For each $a \in G$, the function f_a belonging to G is the function which maps every $b \in G$ to $b \circ a \in G$. It is easy to see that the syntactic monoid given by the f_w with $w \in G^*$ is isomorphic to the original group $(G, \circ)$. ▮

Example 8.11. There is a finite monoid which is not of the form $(G_L, \circ)$ for any language L.

Let $(\{\varepsilon, 0, 1, 2\}, \circ)$ be the semigroup with $\varepsilon \circ \varepsilon = \varepsilon$, $a \circ \varepsilon = \varepsilon \circ a = a$ and $a \circ b = b$

for all $a, b \in \{0, 1, 2\}$.

The set $\{0, 1, 2\}$ generates the given monoid. Consider all words over $\{0, 1, 2\}^*$. Two such words v, w define the same member of the semigroup iff either both v, w are empty (and they represent ε) or both v, w end with the same digit $a \in \{0, 1, 2\}$ (and they represent a).

The monoid is the syntactic monoid of the dfa which has alphabet $\Sigma = \{0, 1, 2\}$, states $Q = \{s, 0, 1, 2\}$, start state s and transition function $\delta(q, a) = a$ for all symbols a and states q. One can easily see that after reading a word wa the automaton is in the state a and initially, after reading the empty word ε, it is in s.

Consider now any language $L \subseteq \Sigma^*$ which would be a candidate for $(G_L, \circ) = (G, \circ)$. Furthermore, assume that $L \neq \emptyset$ and $L \neq \Sigma^*$. Then $(G_L, \circ)$ must contain equivalence classes of the form $\Sigma^* \cdot \{a\}$ with $a \in \Sigma$ and one equivalence class of the form $\{\varepsilon\}$ which belongs to the neutral element of the monoid — note that there is always a neutral element, as the semigroup operation belonging to reading ε does not change any state, so the equivalence class of $\{\varepsilon\}$ will not be not be fusionated with any other one except for the case that there is only one equivalence class Σ^*. Furthermore, the equivalence classes of the form $\Sigma^* \cdot \{a\}$ are due to the fact that for each non-neutral element $b \in G$, b is idempotent and $c \circ b = b$ for all other monoid elements c.

Furthermore, if for $a, b \in \Sigma$ the condition $L(a) = L(b)$ holds then in the minimal automaton of L both a, b lead to the same state: for further $c \in \Sigma$, the transition goes to the state representing the equivalence class $\Sigma^* \cdot \{c\}$. Thus there are besides ε at most two further states in the minimal automaton of L and the corresponding syntactic monoid is one of the following three (with names $0, 1$ used for the monoid elements different from ε):

(a) $(\{\varepsilon, 0, 1\}, \circ)$ with ε being the neutral element and $a \circ 0 = 0$ and $a \circ 1 = 1$ for all group elements a;

(b) $(\{\varepsilon, 0\}, \circ)$ with ε being the neutral element and $a \circ 0 = 0$ for all group elements a;

(c) $(\{\varepsilon\}, \circ)$ with $\varepsilon \circ \varepsilon = \varepsilon$.

These three syntactic monoids are all different from the given one.

Exercise 8.12. *Determine the syntactic monoids $(G_{L_k}, \circ)$ for the following languages L_1 and L_2:*

1. $L_1 = \{0^n 1^m : n + m \leq 3\}$;

2. $L_2 = \{w : w$ *has an even number of* 0 *and an odd number of* $1\}$.

Exercise 8.13. *Determine the syntactic monoids $(G_{L_k}, \circ)$ for the following languages L_3 and L_4:*

1. $L_3 = \{00\}^* \cdot \{11\}^*$;
2. $L_4 = \{0,1\}^* \cdot \{00, 11\}$.

Quiz 8.14. *Determine the syntactic monoids* $(G_{L_5}, \circ)$ *for the languages* $L_5 = \{0\}^* \cdot \{1\}^*$.

Notation. For a semigroup $(G, \circ)$, researchers have found various ways to introduce for it the notion of automaticity. In particular there is the approach of Hodgson [41, 42] and later Khoussainov and Nerode [53] who required that the group elements are represented in some arbitrary way but the full semigroup operation is automatic as a function in both inputs while there is also the approach of Epstein, Cannon, Holt, Levy, Paterson and Thurston [28] who only consider finitely generated groups and semigroups and who furthermore only require that multiplication with constants is automatic; while this is a weakening to the case of automatic groups, they also require that the representatives of the group elements are words over the generators. In order to distinguish these two notions of "automatic semigroups", Hodgson's model will from now on be called "fully automatic" (as the full semigroup operation is automatic) while the more popular model of Epstein, Cannon, Holt, Levy, Paterson and Thurston [28] will just be called "automatic".

Description 8.15: Automatic groups and semigroups [28]. An automatic semigroup is a finitely generated semigroup which is represented by a regular subset $G \subseteq F^*$ such that the following conditions hold:

- G is a regular subset of F^*;
- Each element of the semigroup has exactly one representative in G;
- For each $y \in G$ the mapping $x \mapsto el_G(xy)$ is automatic.

The second condition is a bit more strict than usual; usually one only requires that there is at least one representative per semigroup element and that the relation which checks whether two representatives are equal is automatic. Furthermore, for monoids and groups, unless noted otherwise, ε is used as the representative of the neutral element.

Furthermore, an automatic semigroup $(G, \circ, \varepsilon)$ is called biautomatic if for each $y \in G$ both mappings $x \mapsto el_G(xy)$ and $x \mapsto el_G(yx)$ are automatic functions.

A semigroup $(G', *, \varepsilon)$ is isomorphic to $(G, \circ, \varepsilon)$ iff there is a bijective function $f : G' \to G$ with $f(\varepsilon) = \varepsilon$ and $f(v * w) = f(v) \circ v(w)$ for all $v, w \in G'$. One says that $(G', *, \varepsilon)$ has an automatic representation iff it is isomorphic to a automatic semigroup and $(G', *, \varepsilon)$ has a biautomatic representation iff it is isomorphic to a biautomatic semigroup.

Example 8.16. Let $F = \{\overline{a}, a\}$ and $G = a^* \cup \overline{a}^*$. Then $(G, \circ, \varepsilon)$ is a automatic group with

$$\begin{aligned}
a^n \circ a^m &= a^{n+m}; \\
\overline{a}^n \circ \overline{a}^m &= \overline{a}^{n+m}; \\
a^n \circ \overline{a}^m &= \begin{cases} a^{n-m} & \text{if } n > m; \\ \varepsilon & \text{if } n = m; \\ \overline{a}^{m-n} & \text{if } n < m; \end{cases} \\
\overline{a}^n \circ a^m &= \begin{cases} \overline{a}^{n-m} & \text{if } n > m; \\ \varepsilon & \text{if } n = m; \\ a^{m-n} & \text{if } n < m. \end{cases}
\end{aligned}$$

One can see that the multiplication is realised by an automatic function whenever one of the operands is fixed to a constant. In general, it is sufficient to show that multiplication with the elements in F is automatic.

The additive group of integers $(\mathbb{Z}, +, 0)$ is isomorphic to this automatic group; in other words, this group is a automatic representation of the integers. The group element a^n represents $+n$ (so aaa is $+3$), $\overline{a}^n$ represents $-n$ and ε represents 0. The operation $\circ$ is isomorphic to the addition $+$, for example $a^n \circ \overline{a}^m$ has as result the representative of $n - m$.

Example 8.17. Let $F = \{\overline{a}, a, b, c\}$ generate a monoid with the empty string representing the neutral element, $G = (a^* \cup \overline{a}^*) \cdot \{b, c\}^*$ and assume that the following additional rules governing the group operations: $ba = c$, $ca = ab$, $a\overline{a} = \varepsilon$, $\overline{a}a = \varepsilon$. These rules say roughly that $\overline{a}$ is inverse to a and c is an element representing ba. One can derive further rules like $ab = baa$ and $ba^{2n} = a^n b$.

Now one uses these rules see that the monoid is automatic, where $w \in G$ and n is the number of c at the beginning of w (when multiplying with a) and the number of b at the end of w (when multiplying with $\overline{a}$), the parameters m, n can be any number in $\mathbb{N}$:

$$\begin{aligned}
w \circ b &= wb; \\
w \circ c &= wc; \\
w \circ a &= \begin{cases} a^{m+1}b^n & \text{if } w = a^m c^n; \\ \overline{a}^m b^n & \text{if } w = \overline{a}^{m+1} c^n; \\ vcb^n & \text{if } w = vbc^n; \end{cases} \\
w \circ \overline{a} &= \begin{cases} a^m c^n & \text{if } w = a^{m+1} b^n; \\ \overline{a}^{m+1} c^n & \text{if } w = \overline{a}^m b^n; \\ vbc^n & \text{if } w = vcb^n; \end{cases}
\end{aligned}$$

To see these rules, consider an example: $bcbb \circ \overline{a} = bbabb \circ \overline{a} = bbbaab \circ \overline{a} = bbbabaa \circ \overline{a} = bbbaba = bbcc$, so $vcb^2 \circ a = vbc^2$.

Note that multiplication from the other side is not realised by an automatic function in this representation; indeed, $b \circ a^{2n}bc = a^n bbc$ and this would involve halving the length of the part a^{2n} what is impossible. Indeed, some problem of this type occurs in every automatic representation of this monoid and the monoid does not have a biautomatic representation.

Exercise 8.18. *Consider a monoid $G = a^*b^*$ with the additional rule that $b \circ a = b$ and neutral element ε. Show that this representation is automatic but not biautomatic. Does the monoid have a biautomatic representation?*

Hodgson [41, 42] as well as Khoussainov and Nerode [53] took a more general approach where they did not require that group elements are represented by strings over generators. This representation can be used for all semigroups.

Description 8.19: Automatic groups and semigroups in the framework of Hodgson, Khoussainov and Nerode [41, 42, 53]. A structure $(G, \circ)$ is called a fully automatic semigroup iff G is a regular subset of Σ^* for some finite alphabet Σ, the function $\circ : G \times G \to G$ is an automatic function and $\circ$ satisfies the law of associativity, that is, satisfies $x \circ (y \circ z) = (x \circ y) \circ z$ for all $x, y, z \in G$. A fully automatic monoid / group is a fully automatic semigroup with a neutral element / a neutral element and an inverse for every group element. A semigroup has a fully automatic presentation iff it is isomorphic to a fully automatic semigroup $(G, \circ)$.

Exercise 8.20. *In order to represent $(\mathbb{Z}, +, 0)$, use $\Sigma = \{0, 1, +, -\}$ and use 0 to represent the 0 and $a_0a_1 \ldots a_n+$ with $a_n = 1$ and $a_0, a_1, \ldots, a_{n-1} \in \{0, 1\}$ to represent the positive integer $\sum_{m \leq n} a_m \cdot 2^m$ and $a_0a_1 \ldots a_n-$ with $a_n = 1$ and $a_0, a_1, \ldots, a_{n-1} \in \{0, 1\}$ to represent the negative integer $-\sum_{m \leq n} a_m \cdot 2^m$. Show that now the addition on the so represented group of the integers is an automatic function (with two inputs). Explain why numbers like 000000001+ (256) and 0101+ (10) and 010100001+ (266) are given with the least significant bits first and not last in this presentation.*

Exercise 8.21. *Consider again the monoid $G = a^*b^*$ with the additional rule that $b \circ a = b$ from Exercise 8.18. Now, for $h, k, i, j \in \mathbb{N}$,*

$$a^h b^k \circ a^i b^j = \begin{cases} a^{h+i} b^j & \textit{if } k = 0; \\ a^h b^{k+j} & \textit{if } k > 0; \end{cases}$$

use these rules to find a fully automatic presentation in the sense of Hodgson, Khoussainov and Nerode for this group.

Theorem 8.22. *The strings over a finite alphabet Δ with at least two symbols plus the concatenation form a semigroup which has an automatic representation in the sense of Epstein, Cannon, Holt, Levy, Paterson and Thurston [28] but not a fully automatic representation in the sense of Hodgson, Khoussainov and Nerode.*

Proof. First, one can easily see that Δ^* itself is a regular set in which every string over Δ has a unique representation and in which Δ is the set of generators of the corresponding semigroup; concatenation with a fixed string y is an automatic function mapping x to xy.

Assume now that some regular set $G \subseteq \Sigma^*$ represents the semigroup Δ^* and that $\circ$ represents the concatenation in this semigroup G and that $\circ$ is fully automatic. Now let $F \subseteq G$ be the set of representatives of Δ in G. Let $F_1 = F$ and $F_{n+1} = \{v \circ w : v, w \in F_n\}$. There is a constant c such that each word in F_1 has at most length c and that $v \circ w$ has at most length $\max\{|v|, |w|\} + c$ for all $v, w \in G$. It follows by induction that all words in F_n have at most length cn.

By induction one can see that F_1 represents the strings in Δ^1 and F_n represents the strings in Δ^{2^n}. As Δ has at least 2 elements, there are at least 2^{2^n} members in the set F_n. On the other hand, F_n has at most $\sum_{m \leq nc} |\Sigma|^m$ elements, which can — assuming that $|\Sigma| \geq 2$ — be estimated with $|\Sigma|^{nc+1}$. This gives a contradiction for large n as the upper bound $n \mapsto |\Sigma|^{nc+1}$ is exponential while the lower bound $n \mapsto 2^{2^n}$ is double-exponential. So, for large n, the lower bound is not below the upper bound. Hence the above mentioned fully automatic representation of the monoid of strings over a finite alphabet with at least two letters cannot exist. ▮

Exercise 8.23. *Assume that F is finite and has at least two elements. Let $(G, \circ, \varepsilon)$ be the free group generated by F. Show that this group is not isomorphic to a fully automatic group (in the sense of Hodgson, Khoussainov and Nerode); that is, this group does not have a fully automatic presentation.*

Example 8.24. Let a, b, c be generators of the monoid satisfying $c \circ b = c$ and $b \circ a = a$ and $a \circ c = c \circ a$ which gives the equation

$$a^i b^j c^k \circ a^{i'} b^{j'} c^{k'} = \begin{cases} a^i b^{j+j'} c^{k'} & \text{if } k = 0 \wedge i' = 0; \\ a^{i+i'} b^{j'} c^{k'} & \text{if } k = 0 \wedge i' > 0; \\ a^i b^j c^{k+k'} & \text{if } k > 0 \wedge i' = 0; \\ a^{i+i'} c^{k+k'} & \text{if } k > 0 \wedge i' > 0; \end{cases}$$

where $i, j, k, i', j', k' \in \mathbb{N}$. This monoid is fully automatic. One can represent the group as a convolution of three copies of $(\mathbb{N}, +)$ and use above formula for $\circ$. When adding these components, one has both entries available of the input and those three of the output available. Then one can test for each entry whether they are zero or whether the sum of two entries give a third. This is sufficient to verify that the update is done according to the formula above.

However, the monoid is not automatic. Intuitively, the reason is that when multiplying with c from the front or with a from the back, the corresponding deletions of the entries for b cannot be done. The deletion is needed. Assume that i, j, k are given with j much larger than i, k. Note that $(a^{i-1} \circ b^j \circ c^k) \circ a$ and $a^i \circ c^k$ represent the aame semigroup elements, thus the last multiplication in the first expression must match the long representation of $a^{i-1} \circ b^j \circ c^k$ whose length is linear in j to the shorter representation of $a^i \circ c^k$ whose length is linear in $i+k$. During this shrinking process, the representation of c^k at the end of both words must be moved to the front. This is something what an automatic function can only do for a constant value of k but not when k is a parameter ranging over many values as it is here the case.

Similarly if one would consider multiplication with constants from the front only, then the corresponding representation would also not be automatic, as when multiplying $a^i b^j c^{k-1}$ from the front with c, a deletion operation as indicated above has to be done and the representatives of c^k have moved a lot to the front, what the automatic function cannot do.

Exercise 8.25. *Let a, b be generators of a group satisfying*

$$a^h b^k \circ a^i b^j = \begin{cases} a^{h+i} b^{k+j} & \text{if } k \text{ is even;} \\ a^{h-i} b^{k+j} & \text{if } k \text{ is odd;} \end{cases}$$

where $h, k, i, j \in \mathbb{Z}$. Show that this group is biautomatic as well as fully automatic; find for both results representations.

There is no finitely generated group which is fully automatic in the sense of Hogdson [41, 42], Khoussainov and Nerode [53], but not automatic in the sense of Epstein, Cannon, Holt, Levy, Paterson and Thurston [28]. But for monoids, there is such an example.

Exercise 8.26. *Use the pumping lemma to show the following: If $(G, \circ)$ is a finitely generated semigroup which on one hand has a fully automatic semigroup operation and on the other hand uses natural representatives, that is, all members of G are words over generators from a finite set F of generators, then G is a finite semigroup.*

Note that there are indeed infinite groups which have both an automatic representation in the sense of Epstein, Cannon, Holt, Levy, Paterson and Thurston [28] and a fully automatic representation in the sense of Hodgson [41, 42]. What the above exercise shows is that these two representations have for infinite semigroups necessarily to be distinct, they cannot be the same.

Exercise 8.27. *Construct an automatic representation of the group H of all rationals (positive or negative) with multiplication which have only one-digit prime factors – so they are of the form $2^h \cdot 3^i \cdot 5^j \cdot 7^k$ or $-2^h \cdot 3^i \cdot 5^j \cdot 7^k$ with $h, i, j, k \in \mathbb{Z}$.*

Exercise 8.28. *Construct a fully automatic representation for the group H from Exercise 8.27. Note that one cannot use the same representation by Exercise 8.26.*

Exercise 8.29. *Let a representation $(G, \circ)$ of a group be given and let f map each $x \in G$ to its inverse. Are the following true: (a) If $(G, \circ)$ is automatic then f is automatic; (b) If $(G, \circ)$ is fully automatic then f is automatic?*

For an automatic or fully automatic semigroup $(G, \circ)$, let $f_G(n)$ be the number of elements which have representatives in G of length shorter than n. The next exercise ask for possible values of $f_G(n)$.

Exercise 8.30. *Prove that every automatic representation of $(\mathbb{Z}, +)$ satisfies that the function f_G is bounded by the expression cn for some constant c.*

Exercise 8.31. *Provide an example of an automatic group with quadratic function f_G.*

Exercise 8.32. *Determine the function f_G of the monoid with alphabet $\{a, \overline{a}, b, c\}$ and the rules $ab = baa, ba = c, a\overline{a} = \varepsilon, \overline{a}a = \varepsilon$, where ε is the neutral element and the set of representatives is $G = (\{a\}^* \cup \{\overline{a}\}^*) \cdot \{b, c\}^*$.*

Exercise 8.33. *For fully automatic semigroups, the above questions would be quite easy by showing the folloinwg fact: For every nonempty regular set G there is an Abelian semigroup operation $\circ$ such that $(G, \circ)$ is fully automatic.*

9 Automatic Structures in General

Fully automatic groups (in the sense of Hodgson, Khoussainov and Nerode) are only a special case of an automatic structure. For automatic structures which are neither semigroups nor groups, one just uses the term “automatic” instead of “fully automatic”, as there are no competing concepts with the same name for this type of structure.

Definition 9.1: Automatic Structure [41, 42, 53]. *A structure $(A, R_1, R_2, \ldots, R_h, f_1, f_2, \ldots, f_k)$ is called automatic iff A is a regular subset of some domain Σ^* and all relations $R_1, R_2, \ldots, R_h$ and functions $f_1, f_2, \ldots, f_k$ are automatic. More general, also structures which are isomorphic to automatic structures are called automatic.*

Example 9.2. The automatic structure $(0^*, Succ)$ with $Succ(x) = x0$ is isomorphic to the automatic structure of the natural numbers with successor $Succ(n) = n+1$. In this structure, the addition is not automatic, as the function $n \mapsto n + n$ has in that representation the graph $\{conv(0^n, 0^{2n}) : n \in \mathbb{N}\}$ which is not regular.

An automatic semigroup can be represented as an automatic structure consisting of a regular set R plus functions f_a mapping x to $x \circ a$ for every generator a of the semigroup. So $(\{0,1\}^*, x \mapsto 0x, x \mapsto 1x, \varepsilon)$ is an automatic structure representing the monoid of binary strings with concatenation; here is, however, instead of the full concatenation only the concatenation with the fixed strings 0 and 1 given.

$(\{0,1\}^*, \{0\}^*, <_{ll})$ is the set of binary strings with length-lexicographical order plus an additional set representing all those strings which only consist of 0s. Now $|x| < |y|$ is definable in this structure as

$$|x| < |y| \Leftrightarrow \exists z \in \{0\}^* \, [x <_{ll} z \wedge (z = y \vee z <_{ll} y)].$$

So a string in $\{0\}^*$ is the shortest string among the strings of its length; if one could compare the length of strings, then one could define

$$x \in \{0\}^* \Leftrightarrow \forall y <_{ll} x \, [|y| < |x|].$$

So comparison of sizes and $\{0\}^*$ are definable from each other using the ordering $<_{ll}$.

Description 9.3: Semirings and Rings. A commutative ring $(A, \oplus, \otimes, 0, 1)$ with 1 satisfies the following axioms:

- $\forall x, y \in A \, [x \oplus y = y \oplus x]$;
- $\forall x, y, z \in A \, [x \oplus (y \oplus z) = (x \oplus y) \oplus z]$;
- $\forall x, y \in A \, [x \otimes y = y \otimes x]$;

- $\forall x, y, z \in A\,[x \otimes (y \otimes z) = (x \otimes y) \otimes z]$;
- $\forall x, y, z \in A\,[x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)]$;
- $\forall x \in A\,[x \oplus 0 = x \wedge x \otimes 1 = x]$;
- $\forall x \in A\,\exists y \in A\,[x \oplus y = 0]$.

The laws listed here are the commutative and associative laws for $\oplus$ and $\otimes$, the law of distributivity, the law on the neutral elements and the existence of an inverse with respect to $\oplus$.

If one replaces the last axiom (existence of an inverse for addition) by $x \otimes 0 = 0 \otimes x = 0$ for all x, the structure $(A, \oplus, \otimes, 0, 1)$ is called a commutative semiring with 1. Note that the statement that $x \otimes 0 = 0$ is true in all rings, for semirings one needs to postulate it explicitly.

Furthermore, "commutative" refers to the multiplication $\otimes$. Also for rings which are not commutative, one assumes that the addition $\oplus$ is commutative. A structure without any law of commutativity is called a near-ring or a near-semiring which is defined as follows: Here a near-semiring $(A, \oplus, \otimes, 0)$ satisfies the associative laws for $\oplus$ and $\otimes$, the neutrality of 0 for the addition $\oplus$, one of the distributive laws $(x \oplus y) \otimes z = (x \otimes z) \oplus (y \otimes z)$ or $x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$ and $0 \otimes x = 0$ for all $x, y, z \in A$. A near-ring has the additional property that $(A, \oplus, 0)$ is a group, that is, that every element has an inverse for $\oplus$. Which of the distributive laws is used in a near-ring is a convention; each near-ring satisfying one distributive law can be transformed into a near-ring satisfying the other distributive law (by inverting the order of the operations in the near-ring).

An example for a ring is $(\mathbb{Z}, +, *, 0, 1)$; furthermore, for every $n \in \{2, 3, \ldots\}$, the structure $(\{0, 1, \ldots, n-1\}, +, *, 0, 1)$ with addition and multiplication taken modulo n is a ring. $(\mathbb{Z} \times \mathbb{Z}, +, *, (0, 0), (1, 1))$ with $+$ and $*$ carried out componentwise is a ring.

Example 9.4. Let $(F, +, \cdot)$ be the finite ring of addition and multiplication modulo a fixed number $r \in \{2, 3, \ldots\}$; so the domain F is $\{0, 1, \ldots, r-1\}$ and for $i, j \in F$ one defines that $i \oplus j = i + j$ if $i + j < r$ and $i \oplus j = i + j - r$ if $i + j \geq r$. Furthermore, $i \otimes j$ is the last digit of $i * j$ when written in an r-adic number system.

Now consider in the set $F^{\mathbb{N}}$ all functions $f : \mathbb{N} \to F$ which are eventually constant. Each such function can be represented by a string $a_0 a_1 \ldots a_n \in F^*$ where $f(m) = a_m$ for $m \leq n$ and $f(m) = a_n$ for $m > n$. Furthermore, let $rep(a_0 a_1 \ldots a_n) = a_0 a_1 \ldots a_m$ for the least $m \in \{0, 1, \ldots, n\}$ such that $a_n = a_k$ for all $k \in \{m, m+1, \ldots, n\}$. Let $A = \{rep(a_0 a_1 \ldots a_n) : n \in \mathbb{N} \wedge a_0, a_1, \ldots, a_n \in F\}$. Now one can define the pointwise operations $\oplus$ and $\otimes$ on A as follows:

Given $a_0 a_1 \ldots a_n$ and $b_0 b_1 \ldots b_m$, one says that $c_0 c_1 \ldots c_k = a_0 a_1 \ldots a_n \oplus b_0 b_1 \ldots b_m$ iff for all h, $c_{\min\{k,h\}} = a_{\min\{n,h\}} \oplus b_{\min\{m,h\}}$. Similarly for $\otimes$. These operations correspond to the pointwise addition and multiplication on eventually constant functions in $F^{\mathbb{N}}$. The element 0 represents the neutral element for the addition, that is, the function which is everywhere 0; the element 1 represents the neutral element for the multiplication, that is, the function which is everywhere 1.

The resulting structure is automatic. Furthermore, it is an example of a well-known type of mathematical structures, namely of an infinite ring.

Description 9.5: Orderings. An ordering $\sqsubset$ on a base set A is a relation which satisfies the following two axioms:

- $\forall x, y, z \in A\, [x \sqsubset y \wedge y \sqsubset z \Rightarrow x \sqsubset z]$;
- $\forall x[\neg x \sqsubset x]$.

The first law is called transitivity, the second irreflexivity. An ordering is called linear iff any two $x, y \in A$ are comparable, that is, satisfy $x \sqsubset y$ or $x = y$ or $y \sqsubset x$. Often one writes $x \sqsubseteq y$ for $x \sqsubset y \vee x = y$.

There are well-known automatic orderings, in particular $<_{lex}$ and $<_{ll}$ can be introduced on any set of strings. Also the ordering which says that $x \sqsubset y$ iff x is shorter than y can be introduced on every regular set. Another one $x \preceq y$ says that y extends x, that is, $y = xz$ for some z.

Exercise 9.6. *Consider* $(\{0\} \cdot \{0,1\}^* \cup \{1\}, \max_{lex}, \min_{lex}, 0, 1)$. *Show that this structure is an automatic semiring and verify the corresponding properties as well as the automaticity.*

Does this work for the maximum and minimum of any automatic linear ordering $\sqsubset$ *when the least element* 0 *and greatest element* 1 *exist?*

Given the commutative automatic semiring $(R, +, *, 0, 1)$, *consider the extension on* $R \times R \times R$ *with the componentwise operation* $+$ *and the new multiplication* $\odot$ *given by* $(x, y, z) \odot (x', y', z') = (x * x', y * y', x * z' + z * y')$? *Is this a semiring? Is* $\odot$ *commutative? What are the neutral elements for* $+$ *and* $\odot$ *in this ring? Prove the answer.*

A field is a ring with 1 where there exists for every $x \in A - \{0\}$ an $y \in A$ with $x \otimes y = 1$. The next result shows that infinite automatic fields do not exist.

Theorem 9.7. *There is no infinite automatic field.*

Proof. Assume that $(A, +, *, 0, 1)$ is an infinite automatic field; one can enrich this structure with the length-lexicographic ordering which is also automatic.

Now let $f(x)$ be the length-lexicographically first $y \in A$ such that for all $a, a', b, b' \leq_{ll} x$ with $a \neq a'$ it holds that $(a - a') * y \neq b' - b$. The y exists. To see this, note that for each quadruple (a, a', b, b') with $a - a' \neq 0$ there is at most one y with $(a - a') * y = b' - b$. If there would be a second y' with the same property then $(a - a') * (y - y') = 0$ and now one can derive a contradiction. Let z be the multiplicative inverse of $y - y'$, that is, $(y - y') * z = 1$. Now $((a - a') * (y - y')) * z = 0$ and $(a - a') * ((y - y') * z) = a - a'$. However, by assumption, $a - a' \neq 0$, hence the law of associativity would be violated, a contradiction. Hence each quadruple (a, a', b, b') with $a \neq a'$ disqualifies only one y and as A has infinitely many y, there is some y which can be $f(x)$.

The function f is first-order defined using automatic parameters and therefore automatic.

Now consider $g(x, a, b) = a * f(x) + b$ and $h(x) = \max_{ll}\{g(x, a, b) : a, b \in A \wedge a, b \leq_{ll} x\}$. Both functions are automatic. Furthermore, if there are m elements in A up to x then there are at least m^2 elements up to $h(x)$; the reason is that if $(a, b) \neq (a', b')$ and $a, b \leq_{ll} x$ then one of the following two cases holds: in the case $a = a'$, it holds that $a * f(x) = a' * f(x)$ and $b \neq b'$ and this gives $a * f(x) + b \neq a' * f(x) + b'$; in the case $a \neq a'$, it holds that $(a - a') * f(x) \neq b' - b$ and again $a * f(x) + b \neq a' * f(x) + b'$ by a simple transformation of the inequality. So A has at least m^2 many elements of the form $a * f(x) + b$ with $a, b \leq_{ll} x$ and $a * f(x) + b \leq_{ll} h(x)$.

Now let $k \in \mathbb{N}$ be a constant so large that the field-elements A has at least 2 elements shorter than k and that $|h(x)| \leq |x| + k$ for all $x \in A$. Now let $A_r = \{x \in A : |x| \leq r \cdot k\}$. By induction, A_0 has at least 2^{2^0} elements and A_{r+1} has at least $2^{2^{r+1}}$ elements, as the number of elements in A_{r+1} is at least the square of the number of elements in A_r, hence $|A_{r+1}| \geq |A_r| \cdot |A_r| \geq 2^{2^r} \cdot 2^{2^r} = 2^{2^r + 2^r} = 2^{2^{r+1}}$. This would mean that the function $r \mapsto |A_r|$ grows at least double-exponentially in contradiction to the fact that all strings in A_r have length up to $r \cdot k$ so that there are at most $(1 + |\Sigma|)^{r \cdot k + 1}$ elements in A_r where Σ is the alphabet used. This contradiction shows that a field cannot be automatic. ▮

Description 9.8: Ordinals. In particular of interest are sets of ordinals. Small ordinals — and only they are interesting — can be viewed as expressions using finitely often exponentiation, power and addition and as constants ω and 1, where 1 stands for ω^0. Each exponentiation is of the form ω^α where α is an ordinal defined in the same way. If $\alpha \leq \beta$ then $\omega^\alpha \leq \omega^\beta$. Furthermore, $\alpha < \omega^\alpha$ for all the α considered here. Sums are always written with ω-powers in descending order. For example, $\omega^{\omega+\omega} + \omega^{\omega+1+1} + \omega^{1+1+1+1} + \omega^{1+1+1} + \omega^{1+1+1} + 1 + 1 + 1 + 1$. To simplify notation, repeated additions can be replaced by the corresponding natural numbers and $\omega^\alpha \cdot 5$ abbreviates that one has a sum of 5 identical terms ω^α. Above example gives $\omega^{\omega \cdot 2} + \omega^{\omega+2} + \omega^4 + \omega^3 \cdot 2 + 4$. One can compare two ordinals given by sums by looking at the first ω-power from above which has a different coefficient in both ordinals and then the ordinal with the larger coefficient is the larger one: $\omega^{\omega+2} + \omega^\omega \cdot 3 + \omega^{256} \cdot 373 <$

$\omega^{\omega+2} + \omega^{\omega} \cdot 4 < \omega^{\omega+2} + \omega^{\omega+1} < \omega^{\omega+2} + \omega^{\omega+1} + \omega^{256}$.

There is furthermore an addition of ordinals. Given the ordinals as descending sums of ω-powers, the rule is to write the ω-powers of the second operand behind those of the first and then to remove all powers ω^{α} for which there is a power ω^{β} behind with $\alpha < \beta$. The coefficients of the highest ω-power occurring in the second operand are added in the case that this ω-power occurs in both operands. Here an example: $\omega^8 + \omega^5 + \omega^2$ plus $\omega^5 + \omega^4$ gives $\omega^8 + \omega^5 + \omega^5 + \omega^4 = \omega^8 + \omega^5 \cdot 2 + \omega^4$.

Note that the addition of the ordinals is not commutative. On one hand $\omega + 1 \neq \omega$ and on the other hand $1 + \omega = \omega$ by the cancellation-rule. Furthermore, one can introduce a multiplication for ordinals and show that the resulting structure is a near-semiring. As $(\mathbb{N}, +, \cdot, 0, 1)$ is a substructure of the ordinals whenever they go up to ω or beyond and as that structure is not automatic, the multiplication of ordinals is uninteresting for automatic structures.

So one investigates the ordinals with respect to the automaticity of their ordering and their addition. In particular, for given ordinal α one is interested in the linearly ordered set $\{\beta : \beta < \alpha\}$ and the following question had been investigated for years: For which α is the structure $(\{\beta : \beta < \alpha\}, <)$ isomorphic to an automatic linear order?

Example 9.9: Ordinals [21]**.** The ordinals below ω^4 are automatic. For this, recall that there is an automatic copy of $(\mathbb{N}, +, <)$. One can now represent each such ordinal by $conv(a_0, a_1, a_2, a_3)$ standing for $\omega^3 \cdot a_3 + \omega^2 \cdot a_2 + \omega \cdot a_1 + a_0$. Now the addition follows the following equation:

$$\begin{array}{r} conv(a_0, a_1, a_2, a_3) + \\ conv(b_0, b_1, b_2, b_3) = \end{array} \begin{cases} conv(a_0 + b_0, a_1, a_2, a_3) & \text{if } b_1 = 0,\ b_2 = 0,\ b_3 = 0; \\ conv(b_0, a_1 + b_1, a_2, a_3) & \text{if } b_1 > 0,\ b_2 = 0,\ b_3 = 0; \\ conv(b_0, b_1, a_2 + b_2, a_3) & \text{if } b_2 > 0,\ b_3 = 0; \\ conv(b_0, b_1, b_2, a_3 + b_3) & \text{if } b_3 > 0. \end{cases}$$

This function is automatic, as one can decide with automatic predicates which case applies and then form a convolution of functions which either copy one of the input-components or add two of them. Furthermore, the relation $<$ is first-order definable from the addition:

$$\begin{aligned} & conv(a_0, a_1, a_2, a_3) < conv(b_0, b_1, b_2, b_3) \Leftrightarrow \\ & \quad \exists conv(c_0, c_1, c_2, c_3) \neq conv(0, 0, 0, 0) \\ & \qquad [conv(a_0, a_1, a_2, a_3) + conv(c_0, c_1, c_2, c_3) = conv(b_0, b_1, b_2, b_3)]. \end{aligned}$$

Hence the ordinals below the power ω^4 form a fully automatic monoid with ordering. Delhommé [21] showed the following more general result.

Theorem 9.10: Delhommé's Characterisation of Automatic Ordinals [21]**.** *The following is equivalent for any ordinal α:*

(a) *$\alpha < \omega^\omega$;*
(b) *$(\{\beta : \beta < \alpha\}, <)$ is an automatic structure;*
(c) *$(\{\beta : \beta < \alpha\}, +)$ is an automatic structure.*
Here, in the case of (c), the domain of $+$ is the set of all pairs (β, γ) with $\beta + \gamma < \alpha$; in the case of α being an ω-power, this domain is the set $\{\beta : \beta < \alpha\} \times \{\gamma : \gamma < \alpha\}$.

Proof. Above it was shown for $\alpha = \omega^4$ that the ordinals below α with addition and ordering can be represented as an automatic structure. This proof generalises to all ω^n. Furthermore, if $\alpha \leq \omega^n$ for some $n \in \mathbb{N}$, then the set $\{\beta : \beta < \alpha\}$ can be defined in the automatic structure $(\{\beta : \beta < \omega^n\}, +, <)$ using α as a parameter and the resulting structure can serve to satisfy **(b)** and **(c)**, simultaneously. So assume by way of contradiction that $\alpha \geq \omega^\omega$ and that the structure $(\{\beta : \beta < \alpha\}, <)$ is automatic with a regular domain A and an automatic relation $<$. Add the string extension relation $\preceq$ on the domain to this structure as well as a relation to compare the length of strings. Assume that u_n represents ω^n in this structure. Now consider for all strings v with $|v| = |u_n|$ the sets $B_{u_n,v} = \{w \in A : v \preceq w \wedge w < u_n\}$.

Each set $B_{u_n,v}$ is defined in an uniform way and there is a finite automaton checking whether $w \in B_{u_n,v}$ when reading $conv(u_n, v, w)$. Furthermore, for $w_1, w_2 \in B_{u_n,v}$, there is a further automaton checking whether $w_1 < w_2$ in a way uniform in $conv(u_n, v, w_1, w_2)$. It is now easy to see that the structure $B'_{u_n,v} = (\{w' : vw' \in B_{u_n,v}\}, <')$ with $w'_1 <' w'_2 \Leftrightarrow vw_1 < vw_2$ only depends on the states of the automata recognising $B_{u_n,v}$ and $<$ after having read the first $|u_n|$ symbols of $conv(u_n, v, vw')$ and $conv(u_n, v, vw'_1, vw'_2)$, respectively. Hence there are only finitely many different such structures.

However, the set $\{w : w < u_n\}$ is for each u_n the union of finitely many sets $B_{u_n,v}$ and a finite set (of ordinals represented by strings shorter than u_n). One of the partially ordered sets $(B_{u_n,v}, <)$ must be isomorphic to $(\{\beta : \beta < \omega^n\}, <)$ and therefore the same holds for $(B'_{u_n,v}, <')$. As there are infinitely many such sets (different ordinals give non-isomorphic linearly ordered sets), this is a contradiction to the assumption that the structure of the ordinals below α is automatic. ▮

Exercise 9.11. *The above proof used one fact implicit: If $\{\beta : \beta < \omega^n\}$ is the union of finitely many sets $A_1, A_2, \ldots, A_m$ then one of the sets satisfies that $(A_k, <)$ is isomorphic to $(\{\beta : \beta < \omega^n\}, <)$. For $n = 1$ this is easily to be seen: Every infinite subset of the ordinals below ω is isomorphic to $(\mathbb{N}, <)$ and hence isomorphic to the set of ordinals below ω. For $n = 2$, this can be seen as follows: For each set A_k let*

$$\tilde{A}_k = \{i : \exists^\infty j\, [\omega \cdot i + j \in A_k]\}.$$

Each $i \in \mathbb{N}$ must appear in at least one set $\tilde{A}_k$. Hence there is a k for which $\tilde{A}_k$ is infinite. Now do the following: First, complete the case $n = 2$ by showing that for *each k with $\tilde{A}_k$ being infinite the linearly ordered set $(A_k, <)$ is isomorphic to the set of ordinals below ω^2; second, generalise the whole proof to all $n \in \{3, 4, 5, 6, \ldots\}$.*

Remarks 9.12. There are quite many structures for which one was able to show that they are not automatic structures (that is, not fully automatic monoids): the multiplicative monoid of the natural numbers, of the rationals, of the positive rationals and so on. Furthermore, the polynomial ring in one or more variables over a finite field is not an automatic structure.

It had also been investigated when a graph can be automatic, that is, be isomorphic to an automatic graph. Here (V, E) is an automatic graph iff V is a regular set and the set of edges E on V is an automatic relation. A graph on an infinite and countable set V is called a random graph iff the graph is undirected (that is, $(x, y) \in E \Leftrightarrow (y, x) \in E$ for all $x, y \in V$) and for any disjoint finite sets A, B there is a node x with $(x, y) \in E$ for all $y \in A$ and $(x, y) \notin E$ for all $y \in B$. It is known that all random graphs are isomorphic, that is, if (V, E) and (V', E') are random graphs then there is a bijection $f : V \rightarrow V'$ with $\forall x, y \in V\, [(x, y) \in E \Leftrightarrow (f(x), f(y)) \in E']$. Delhommé showed that no random graph is automatic [21]. The next paragraphs give a proof of this result.

Assume that (V, E) would be an automatic copy of the random graph. Furthermore, let $<_{ll}$ be the automatic length-lexicographic order on V.

Now define a relation $R(x, y)$ as there is no $z <_{ll} y$ with $\forall u \leq_{ll} x\, [(u, z) \in E \Leftrightarrow (u, y) \in E]$. This relation says that y is the first of all nodes z which connect to the nodes up to x in the same way as y. Note that for every x there are only finitely many y with $R(x, y)$, as there are only finitely many ways how a node can connect with the finitely many nodes up to x. As R is first-order defined with automatic parameters, R is automatic. Furthermore, $f_R(x) = max_{ll}\{y : R(x, y)\}$ is an automatic function; thus there is constant c with $|f_R(x)| \leq |x| + c$ for all x and $c \geq |min_{ll}(V)|$.

Let $g(n)$ be the number of element of V up to length $c \cdot n$. Now $g(1) \geq 1$ and $g(n + 1) \geq 2^{g(n)}$, as for each splitting (A, B) of the elements of V up to length $c \cdot n$ there is an y connecting to those in A and not to those in B. On one hand $g(n)$ grows superexponentially and on the other hand there are only exponentially many elements up to length $c \cdot n$. Due to this contradiction, (V, E) cannot be automatic.

Exercise 9.13. *Let $(G, \circ)$ be a fully automatic group and F be a regular subset of G. Is the graph (G, E) with $E = \{(x, y) : \exists z \in F\, [x \circ z = y]\}$ automatic?*

To which extent can the result be transferred to automatic groups? Consider the special cases for F being finite and F being infinite. In which cases are there automatic groups $(G, \circ)$ in the sense of Epstein, Cannon, Holt, Levy, Paterson and Thurston [28] such that for given finite F the graph (G, E) is automatic? How about infinite F?

Exercise 9.14. *Consider the following structure: For $a = (a_0, a_1, \ldots, a_n) \in \mathbb{N}^{n+1}$, let*

$$f_a(x) = \sum_{m=0}^{n} a_m \cdot \binom{x}{m}$$

and let F be the set of all so defined f_a (where n is not fixed). For which of the following orderings $<_k$ is $(F, <_k)$ an automatic partially ordered set?

(1) $a <_1 b \Leftrightarrow f_a \neq f_b$ and $f_a(x) < f_b(x)$ for the first x where they differ;
(2) $a <_2 b \Leftrightarrow \exists^\infty x\, [f_a(x) < f_b(x)]$;
(3) $a <_3 b \Leftrightarrow \forall^\infty x\, [f_a(x) < f_b(x)]$.

Give reasons for the answer.

For the following exercises, let the binary string $val(a_0 a_1 \ldots a_n)$ denote $\sum_m 2^m \cdot a_m$ where $a_m \in \{0, 1\}$ and allow leading zeroes. For convolutions, there is in this specific case no need to distinguish # and 0.

Exercise 9.15. *Construct a two-state dfa which checks whether $val(x) \leq val(y)$ for binary strings x, y.*

Exercise 9.16. *Construct a dfa which checks whether $val(x) < val(y) + val(z)$ for binary strings x, y, z.*

Exercise 9.17. *Construct a dfa which checks whether*

$$\max\{val(x), val(y)\} \leq val(z) + val(z)$$

for binary strings x, y, z.

Exercise 9.18. *Construct a dfa which checks for binary strings x, y, z whether* $\max\{val(x), val(y)\} \leq \min\{val(y), val(z)\}$.

Exercise 9.19. *The structure $(\{0, 1\}^*, Pal, u \mapsto u0, u \mapsto u1)$ is not automatic in the current representations, as the set Pal of all palindromes is not regular. Is there any other automatic presentation of this structure? Prove the answer.*

Yuri Matiyasevich [59] showed that there is a polynomial $p(x, y_1, \ldots, y_9)$ with integer coefficients such that one cannot decide whether for given $x \in \mathbb{N}$ one can find $y_1, \ldots, y_9 \in \mathbb{N}$ with $p(x, y_1, \ldots, y_9) = 0$.

Exercise 9.20. *Show that the ring $(\mathbb{Z}, +, \cdot, <, 0, 1)$ is not automatic.*

Exercise 9.21. *Show that the structure* $(\mathbb{Z}, +, S, <, 0, 1)$ *is not automatic, where* S *is the set of square numbers.*

Exercise 9.22. *Call a subset* $A \subseteq \mathbb{N}$ *eventually* k*-periodic, iff there are* i, j *with* $1 \leq j \leq k$ *such that, for all* $x \geq i$, $A(x) = A(x + j)$. *Prove that for each* $k \in \mathbb{N}$ *with* $k > 0$ *there is an automatic representation of all eventually* k*-periodic sets such that union, intersection and symmetric difference are fully automatic.*

Exercise 9.23. *Call a function* $f : \mathbb{Z} \to \mathbb{Z}$ *to be a* k*-step function iff there are at most* k *vaues* x *with* $f(x) \neq f(x + 1)$. *Construct an automatic structure of all* k*-step functions which has a two-place automatic function* $e, x \to f_e(x)$ *mapping* $x \in \mathbb{Z}$ *to the value* $f_e(x)$ *for the* e*-th member of this class* F_k.

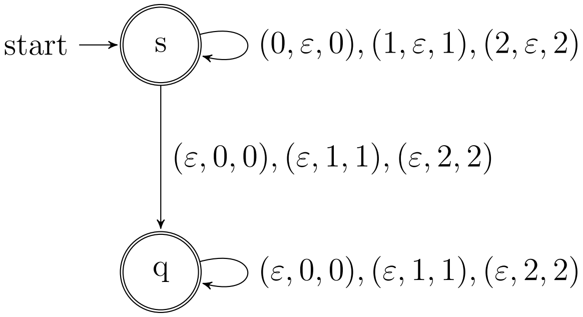
Exercise 9.24. *Prove that one can define* F_k, F_{2k} *from Exercise 9.23 such that there is an automatic function* g_k *mapping each two indices* i, j *for functions in* F_k *to an index* $g_k(i, j)$ *for a function in* F_{2k} *with* $\forall x \, [f_{g_k(i,j)}(x) = f_i(x) + f_j(x)]$.

10 Transducers and Rational Relations

There are two ways to generalise regular sets to regular relations: One is the notion of an automatic relation where all inputs are processed at the same speed (and exhausted shorter inputs padded with #) and the other notion is that of a rational relation which is defined below, where different inputs can be processed at different speed; rational relations are also called asynchronously automatic relations.

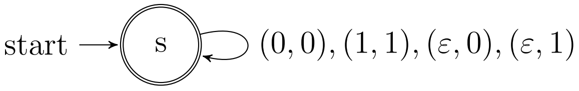
Definition 10.1: Rational Relation. *A relation $R \subseteq (\Sigma^*)^n$ is given by an non-deterministic finite state machine which can process n inputs in parallel and does not need to read them in the same speed. Transitions from one state p to a state q are labelled with an n-tuple $(w_1, w_2, \ldots, w_n)$ of words $w_1, w_2, \ldots, w_n \in \Sigma^*$ and the automaton can go along this transition iff for each input k the next $|w_k|$ symbols in the input are exactly those in the string w_k (this condition is void if $w_k = \varepsilon$) and in the case that the automaton goes on this transition, $|w_k|$ symbols are read from the k-th input word. A word $(x_1, x_2, \ldots, x_n)$ is in R iff there is a run of the machine which ends up in an accepting state after having reached the end-positions of all n words.*

Example 10.2: String Concatenation. The concatenation of strings over Σ^* is a rational relation. The following machine is given for $\Sigma = \{0, 1, 2\}$ and works for other alphabets correspondingly.


start
s
$(0,\varepsilon,0),(1,\varepsilon,1),(2,\varepsilon,2)$
$(\varepsilon,0,0),(\varepsilon,1,1),(\varepsilon,2,2)$
q
$(\varepsilon,0,0),(\varepsilon,1,1),(\varepsilon,2,2)$

In the following graphical notation of a run on three input-words, the state is written always at that position which separates the read and not yet read parts of the input-words; the triple of input-words is $(01, 210, 01210)$ and the run is $(s01, s210, s01210) \Rightarrow (0s1, s210, 0s1210) \Rightarrow (01s, s210, 01s210) \Rightarrow (01q, 2q10, 012q10) \Rightarrow (01q, 21q0, 0121q0) \Rightarrow (01q, 210q, 01210q)$.

Example 10.3: Subsequence-Relation. A string x is a subsequence of y iff it can be obtained by from y by deleting symbols at some positions. The following one-state automaton recognises this relation for the binary alphabet $\{0, 1\}$.

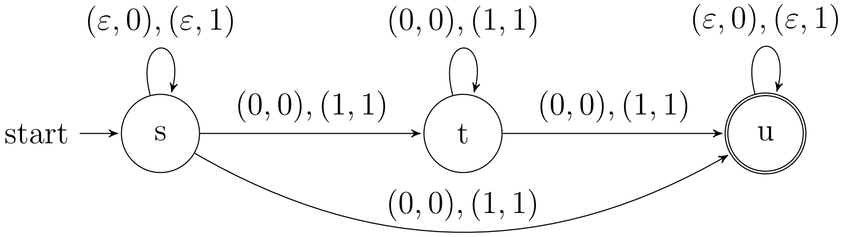


In general, there are one initial accepting state s with self-loops $s \to s$ labelled with (ε, a) and (a, a) for all $a \in \Sigma$.

So if $x = 0101$ and $y = 00110011$ then the automaton goes from s to itself with the transitions $(0,0),(\varepsilon,0),(1,1),(\varepsilon,1),(0,0),(\varepsilon,0),(1,1),(\varepsilon,1)$ and has afterwards exhausted both, x and y. As it is in an accepting state, it accepts this pair.

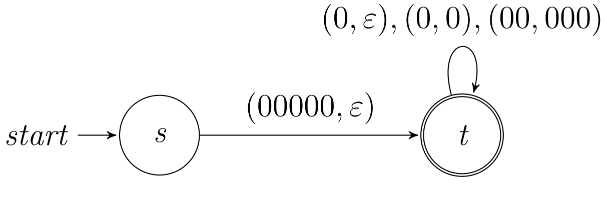
However, if $x = 00111$ and $y = 010101$ then the automaton cannot accept this pair: it gets stuck when processing it. After the first $(0,0)$, it has to use the transition $(\varepsilon,1)$ in order to go on and can afterwards use the transition labelled $(0,0)$ again. But once this is done, the automaton has now on the x-side of the input 111 and on the y-side 101 so that it could go on with using $(1,1)$ once and would then have to use $(\varepsilon,0)$ and afterwards $(1,1)$ again. However, now the automaton gets stuck with 1 being on the x-side while the y-side is exhausted. This is indeed also correct this way, as x is not a subsequence of y.

Example 10.4. This rational relation recognises that x is a non-empty substring of y, that is, $x \neq \varepsilon$ and $y = vxw$ for some $v, w \in \{0,1\}^*$. The automaton is the following.


$(\varepsilon,0),(\varepsilon,1)$
$(0,0),(1,1)$
$(\varepsilon,0),(\varepsilon,1)$
$(0,0),(1,1)$
$(0,0),(1,1)$
start
s
t
u
$(0,0),(1,1)$

When the automaton is in s or u, it parses the parts of x which are not in y while when going forward from s to u with perhaps cycling in t, the automaton compares x with the part of y which is equal to it in order to verify that x is a subword of y; furthermore, the automaton can do this only if x contains at least one symbol.

Exercise 10.5. *Rational relations got their name, as one can use them in order to express relations between the various inputs words which are rational. For example, one can look at the set of all (x, y) with $|x| \geq \frac{2}{3}|y| + 5$. This relation could be recognised by the following automaton (assuming that $x, y \in \{0\}^*$):*



Make automata which recognise the following relations:

(a) $\{(x, y) \in (0^*, 0^*) : 5 \cdot |x| = 8 \cdot |y|\}$*;*

(b) $\{(x, y, z) \in (0^*, 0^*, 0^*) : 2 \cdot |x| = |y| + |z|\}$*;*

(c) $\{(x, y, z) \in (0^*, 0^*, 0^*) : 3 \cdot |x| = |y| + |z| \vee |y| = |z|\}$*.*

Which automaton needs more than one state?

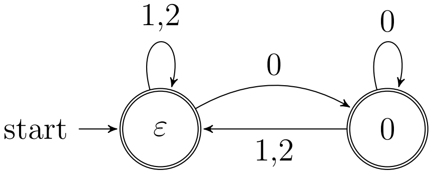
Description 10.6: Transducers. A rational function f mapping strings over Σ to strings over Σ is a function for which there is a rational relation R such that for each $x, y \in \Sigma^*$, $(x, y) \in R$ iff $x \in dom(f)$ and $f(x) = y$. Transducers are mechanisms to describe how to compute such a rational function and there are two types of them: Mealy machines and Moore machines. Both define the same class of functions.

A Mealy machine computing a rational function f is a nondeterministic finite automaton such that each transition is attributed with a pair (v, w) of strings and whenever the machine follows a transition $(p, (v, w), q)$ from state p to state q then one says that the Mealy machine processes the input part v and produces the output part w. If some run on an input x ends up in an accepting state and produces the output y, then every run on x ending up in an accepting state produces the same output and $f(x) = y$; if no run on an input x ends up in an accepting state then $f(x)$ is undefined.

Every automatic function is also a rational function and computed by a transducer, but not vice versa. For example, the function π preserving all symbols 0 and erasing the symbols $1, 2$ is given by the following one-state transducer: Starting state and accepting state is s, the transitions are $(s, (0, 0), s)$, $(s, (1, \varepsilon), s)$, $(s, (2, \varepsilon), s)$. This function π is not automatic.

Description 10.7: Moore machines [66]**.** Edward Moore [66] formalised functions computed by transducers by the concept of an automaton which is now known as a Moore machine. This is a nondeterministic finite automaton with possibly several starting states such that each state q owns a word w_q and each transition if of the form (q, a, p) for states q, p and elements $a \in \Sigma$. On input $a_1 a_2 \ldots a_n$, an accepting run is a sequence $(q_0, q_1, \ldots, q_n)$ of states starting with a starting state q_0 and ending in an accepting state q_n such that the transition-relation of the nfa permits for each $m < n$ to go from q_m to q_{m+1} on symbol a_{m+1} and the output produced by the run is the word $w_{q_0} w_{q_1} \ldots w_{q_n}$. A function f is computed by a Moore machine iff for each $x \in dom(f)$ there is an accepting run on input x with output $f(x)$ and for each string x and accepting run on input x with output y it holds that $f(x)$ is defined and $f(x) = y$.

First, consider the projection π from $\{0, 1, 2\}^*$ to $\{0\}^*$ which erases all $1, 2$ and preserves all 0; for example, $\pi(012012) = 00$. It needs a Moore machine having two states.



Second, let $f(a_1a_2 \ldots a_n) = 012a_1a_1a_2a_2 \ldots a_na_n012$. That is f doubles each symbol and places 012 before and after the output. The Moore machine given by the following table computes f:

state	starting	acc/rej	output	succ on 0	succ on 1	succ on 2
s	yes	rej	012	p, p'	q, q'	r, r'
p	no	rej	00	p, p'	q, q'	r, r'
q	no	rej	11	p, p'	q, q'	r, r'
r	no	rej	22	p, p'	q, q'	r, r'
s'	yes	acc	012012	–	–	–
p'	no	acc	00012	–	–	–
q'	no	acc	11012	–	–	–
r'	no	acc	22012	–	–	–

Now $f(0212)$ has the accepting run $sprqr'$ and this accepting run produces the output $w_sw_pw_rw_qw_{r'} = 012001100012$. The nondeterminism mainly stems from the fact that the automaton does not know when the last symbol is read; therefore, it has nondeterministically choose between the states and their primed versions: s versus s', p versus p', q versus q' and r versus r'.

For an example with a more severe amount of nondeterminism, consider the function g given by $g(a_1a_2 \ldots a_n) = (\max(\{a_1, a_2, \ldots, a_n\}))^n$, so $g(\varepsilon) = \varepsilon$, $g(000) = 000$, $g(0110) = 1111$ and $g(00512) = 55555$. Now the Moore machine has to produce output in each state, but it has to choose in the first state which output to produce. So one has a starting state s with $w_s = \varepsilon$ and for each symbol a two states q_a and r_a with $w_{q_a} = w_{r_a} = a$. The states s and r_a are accepting, the states q_a are rejecting. The following transitions exist: (s, b, q_a) for all a, b with $b < a$, (s, a, q_a) for all a, (q_a, b, q_a) for all a, b with $b < a$ and (r_a, b, r_a) for all a, b with $b \leq a$. So when the Moore machine sees the first symbol and that is a 0, it has to decide which symbol a to write and there is no way to avoid this nondeterminism.

Exercise 10.8. *Write down Mealy machines for the functions f and g from Description 10.7 of the Moore machines. For both, the alphabet can be assumed to be $\{0, 1, 2\}$.*

Exercise 10.9. *Determine the minimum number m such that every rational function can be computed by a nondeterministic Moore machine with at most m starting states. Give a proof that the number m determined is correct.*

Exercise 10.10. *Say that a Moore machine / Mealy machine is deterministic, if it has exactly one starting state and for it always reads one symbol from the input and for each state and each input symbol it has at most one transition which applies.*

Make a deterministic Moore machine and make also a deterministic Mealy machine which do the following with binary inputs: As long as the symbol 1 *appears on the input, the symbol is replaced by* 0*; if at some time the symbol* 0 *appears, it is replaced by* 1 *and from then onwards all symbols are copied from the input to the output without a change.*

So the function f computed by these machines satisfies $f(110) = 001$, $f(1111) = 0000$, $f(0) = 1$ and $f(110011) = 001011$.

Exercise 10.11. *Let the alphabet be $\{0, 1, 2\}$ and let $R = \{(x, y, z, u) : u$ has has $|x|$ many* 0*s, $|y|$ many* 1*s and $|z|$ many* 2*s$\}$. Is R a rational relation? Prove the result.*

Theorem 10.12 [68]**.** *Assume that $\Sigma_1, \Sigma_2, \ldots, \Sigma_m$ are disjoint alphabets. Furthermore, let π_k be the function which preserves all symbols from Σ_k and erases all other symbols. Then a relation $R \subseteq \Sigma_1^* \times \Sigma_2^* \times \ldots \times \Sigma_m^*$ is rational iff there is a regular set P over a sufficiently large alphabet such that $(w_1, w_2, \ldots, w_n) \in R \Leftrightarrow \exists v \in P\, [\pi_1(v) = w_1 \wedge \pi_2(v) = w_2 \wedge \ldots \wedge \pi_m(v) = w_m]$.*

Proof. First, assume that a nondeterministic finite automaton recognises the rational relation R. Let Q be the set of states of this finite automaton and assume that Q is disjoint to all alphabets Σ_k. Furthermore, let the word $q_0 w_{1,1} w_{1,2} \ldots w_{1,m} q_1 w_{2,1} w_{2,2} \ldots w_{2,m} q_2 \ldots w_{n,1} w_{n,2} \ldots w_{n,m} q_n$ be in P iff q_0 is a starting state, q_n is an accepting state and for each $k < n$ the automaton goes from q_k on $(w_{k+1,1}, w_{k+1,2}, \ldots, w_{k+1,m})$ to q_{k+1}. In other words, P consists of representations of all accepting runs of the nfa on some input and if $v \in P$ then the input-tuple processed in this run is $(\pi_1(v), \pi_2(v), \ldots, \pi_m(v))$.

Second, for the converse direction, assume that a regular language P is given and that the dfa with starting state s and accepting states F is recognising P. Let Q be its states. Now one can make an nfa recognising the relation R by replacing every transition (p, a, q) of the original dfa with $(p, (\pi_1(a), \pi_2(a), \ldots, \pi_m(a)), q)$ in the new nfa. One has now to show that the new nfa recognises exactly the relation R.

Assume that there is a word $v = a_1 a_2 \ldots a_n \in P$ with $(w_1, w_2, \ldots, w_m) = (\pi_1(v), \pi_2(v), \ldots, \pi_m(v))$. There is a run $q_0 a_1 q_1 a_2 \ldots a_n q_n$ of the dfa which accepts the word v. Now one translates this run into $q_0\ (\pi_1(a_1), \pi_2(a_1), \ldots, \pi_m(a_1))\ q_1\ (\pi_1(a_2), \pi_2(a_2), \ldots, \pi_m(a_2))\ q_2 \ldots (\pi_1(a_n), \pi_2(a_n), \ldots, \pi_m(a_n))\ q_n$ and one can see that this is an accepting run of the nfa. Hence $(w_1, w_2, \ldots, w_n) \in R$.

Assume that $(w_1, w_2, \ldots, w_n) \in R$. Then the nfa accepts $(w_1, w_2, \ldots, w_n)$. This acceptance is witnessed by a run of the form q_0 $(\pi_1(a_1), \pi_2(a_1), \ldots, \pi_m(a_1))$ q_1 $(\pi_1(a_2), \pi_2(a_2), \ldots, \pi_m(a_2))$ q_2 $\ldots$ $(\pi_1(a_n), \pi_2(a_n), \ldots, \pi_m(a_n))$ q_n where q_0 is a starting state and q_n is an accepting state and the tuples between two states indicate the symbols read from the corresponding inputs. Then corresponds to an accepting run q_0 a_1 q_1 a_2 q_2 $\ldots$ a_n q_n on the original dfa which then accepts the word $v = a_1 a_2 \ldots a_n$. Hence, $v \in P$ and $(w_1, w_2, \ldots, w_m) = (\pi_1(v), \pi_2(v), \ldots, \pi_m(v))$. ▮

Remark 10.13. Above Theorem of Nivat can also be stated in a more general form. Recall that a homomorphism is a mapping which preserves concatenation and maps every symbol to a finite word. Now an n-ary relation R is rational iff there are n homomorphisms $\pi_1, \ldots, \pi_n$ such that for each symbol at most one of them maps it to a nonempty word and there is a regular set P such that

$$\forall x_1, \ldots, x_n \, [R(x_1, \ldots, x_n) \Leftrightarrow \exists y \in P \, [\pi_1(y) = x_1 \wedge \ldots \wedge \pi_n(y) = x_n]].$$

This allows to use pumping lemmas to show that certain relations are not rational. For the ease of notation, the two letters are different in the next example, so that the Theorem of Nivat applies in its original form.

The relation $R = \{(0^n, 1^{n^2}) : n \geq 1\}$ is not rational.

To see this, one uses the Theorem of Nivat and considers a regular set P such that for each n there is a word $y \in P$ with $0^n = \pi_1(y)$ and $1^{n^2} = \pi_2(y)$.

As the set is regular, it satisfies the block pumping lemma with a constant k. There is an n which is large enough so that $n^2 > (k+1) \cdot (n+1)$. Thus there are at least $k+1$ many 1 without a 0 between them in any word $y \in P$ with $0^n = \pi_1(y)$ and $1^{n^2} = \pi_2(y)$. Thus one can cut the word into blocks such that all inner blocks contain each at least one 1 and no 0.

Now when one pumps up with the block pumping lemma, the number of 1 increases while the number of 0 remains the same. The pumping destroys R and R is not rational.

Description 10.14: Rational Structures. One could replace the requirement that relations are automatic by the requirement that relations are rational in oder to obtain a notion of rational structures. These are more general than automatic structures, but here various properties of automatic structures are lost:

- There are relations and functions which are first-order definable from rational relations without being a rational relation;
- There is no algorithm to decide whether a given first-order formula on automatic relations is true.

So the counterpart of the Theorem of Khoussainov and Nerode on automatic structures does not exist. While some properties are lost, the expressibility is in general higher. So various structures which are not automatic can be represented using rational relations. One example is given above: The monoid given by concatenation of strings over the alphabet $\{0,1\}$.

Post's Correspondence Problem allows to define a rational structure where one cannot decide the first-order theory with an algorithm which uses the automata describing a specific instance as input. In the current context, an instance of Post's Correspondence Problem consists of two homomorphism f, g – given as transducers – which maps words of an index set Σ^* to words over Σ^* (as long as there are at least two indices, what is needed for making the structure interesting, one can use the same alphabet for the indices and the words over it). So the homomorphism f is given by $f(a)$ for all $a \in \Sigma$ and maps a word $a_1 a_2 \ldots a_n \in \Sigma^*$ to $f(a_1) \cdot f(a_2) \cdot \ldots \cdot f(a_n)$. Similarly for g. It is easy to see that every homomorphism is rational: The corresponding transducer has a single state s which is the starting state and accepting; there are transitions are from s to s which are labelled $(a, f(a))$ for each $a \in \Sigma$. Now the rational structure also has equality and one considers the following first-order formula:

$$\exists u \in \Sigma^* \, [u \neq \varepsilon \wedge f(u) = g(u)].$$

This formula is true iff the instance (Σ, f, g) of Post's Correspondence Problem has a solution. As Post's Correspondence Problem is undecidable, the corresponding class of structures does not have a decidable first-order theory.

Nicer would it of course to have a single structure with an undecidable first-order theory. This is indeed possible and the structure is a quite easy one: $(\{0,1\}^*, \cdot, \prec, 0, 1, \varepsilon)$. This is the structure of all binary words with concatenation $\cdot$ and strict prefix-relation $\prec$ as well as the constants for the empty word and the two single-letter words. The result that this theory is undecidable is well-known. Recent work by Kristiansen and Murwananshyaka [56] analyses which types of first-order formulas are already undecidable and finds that the set of all existentially quantified formulas with additional bounded quantifiers is undecidable while the usage of only bounded quantifiers leads to a decidable fragment of the first-order theory of this structure.

Exercise 10.15. *There is a rational representation of the random graph. Instead of coding (V, E) directly, one first codes a directed graph (V, F) with the following properties:*

- *For each $x, y \in V$, if $(x, y) \in F$ then $|x| < |y|/2$;*
- *For each finite $W \subseteq V$ there is a y with $\forall x \, [(x, y) \in F \Leftrightarrow x \in W]$.*

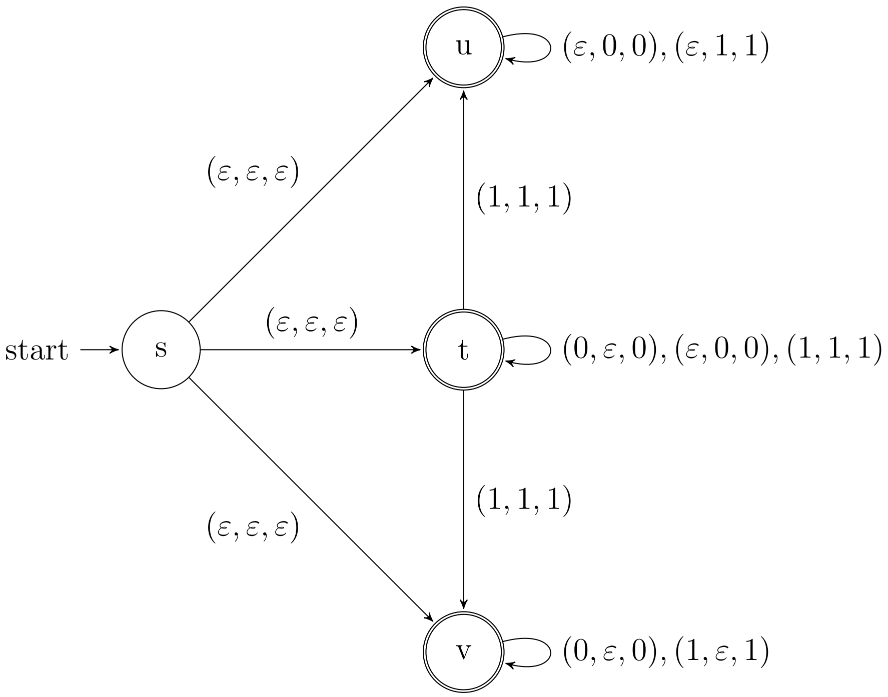
This is done by letting $V = \{00, 01, 10, 11\}^+$ and defining that $(x, y) \in F$ iff there are n, m, k such that $y = a_0 b_0 a_1 b_1 \ldots a_n b_n$ and $a_m = a_k = 0$ and $a_h = 1$ for all h with

$m < h < k$ and $x = b_m b_{m+1} \ldots b_{k-1}$. Give a transducer recognising F and show that this F satisfies the two properties above.

Now let $(x,y) \in E \Leftrightarrow (x,y) \in F \vee (y,x) \in F$. Show that (V,E) is the random graph by constructing to any given disjoint finite sets of strings A, B a string y longer than every string in A and B satisfying that for all $x \in A \cup B$, $(x,y) \in E$ iff $(x,y) \in F$ iff $x \in A$.

Example 10.16. The multiplicative monoid $(\mathbb{N} - \{0\}, *, 1)$ has a rational representation. Note that every natural number is given by its primefactors: So $(n_1, n_2, \ldots, n_k)$ with $n_k > 0$ represents the number $2^{n_1} * 3^{n_2} * \ldots * p_k{}^{n_k}$ and the empty vector represents 1. So 36 is represented by $(2,2)$ (for $2^2 * 3^2$) and 3 is represented by $(0,1)$. Now one has that $36 * 3$ is represented by $(2,3)$ which is the componentwise sum of $(2,2)$ and $(0,1)$. Furthermore, 30 is represented by $(1,1,1)$ so that $36 * 30$ needs that one adjust the length of the shorter vector before one does the componentwise addition: $36 * 30$ is represented by $(2,2) + (1,1,1) = (2,2,0) + (1,1,1) = (3,3,1)$. In other words the set $\mathbb{N} - \{0\}$ with multiplication and the finite vectors of natural numbers with a non-zero last component with the operation of componentwise addition (where 0 is invoked for missing components) are isomorphic monoids.

In the next step, one has to represent each vector $(n_1, n_2, \ldots, n_k)$ by a string, so one takes $0^{n_1} 1 0^{n_2} 1 \ldots 0^{n_k} 1$ and represents the empty vector (standing for the natural number 1) by ε. Now $36 * 3 = 108$ is represented by $001001 * 101 = 0010001$ and $36 * 30 = 1080$ is represented by $001001 * 010101 = 0001000101$. The domain is $\{\varepsilon\} \cup \{0,1\}^* \cdot \{01\}$ and is therefore regular. The following automaton recognises the graph of the rational relation $*$:


u
$(\varepsilon, 0, 0), (\varepsilon, 1, 1)$
$(\varepsilon, \varepsilon, \varepsilon)$
$(1, 1, 1)$
start
s
$(\varepsilon, \varepsilon, \varepsilon)$
t
$(0, \varepsilon, 0), (\varepsilon, 0, 0), (1, 1, 1)$
$(1, 1, 1)$
$(\varepsilon, \varepsilon, \varepsilon)$
v
$(0, \varepsilon, 0), (1, \varepsilon, 1)$

When verifying that $x * y = z$, t is the node which is used as long as x and y are both not exhausted; u is the node to be used when the end of x is reached while the end of y has still to be found while v is the node to be used when the end of y has been reached while the end of x has still to be found. There are transitions on empty tuples from s to all of these three nodes as the two extreme cases that one of x and y is ε need to be covered as well.

For the following exercises, given a a binary rational relation R and a ternary rational relation S and any languages L and H, let $R(L) = \{v : \exists w \in L\,[R(v, w)]\}$ and $S(L, H) = \{u : \exists v \in L\, \exists w \in H\,[S(u, v, w)]\}$.

Exercise 10.17. *Show that if L and H are regular, so are $R(L)$ and $S(L, H)$.*

Exercise 10.18. *Show that if L is context-free, so is $R(L)$.*

Exercise 10.19. *If L, H are context-free, is then also $S(L, H)$ context-free? Prove the answer.*

Exercise 10.20. *If L is context-sensitive, is then also $R(L)$ context-sensitive?*

Exercise 10.21. *For which Boolean operations (union, intersection, set difference, symmetric difference) is there a rational relation S such that $S(L, H)$ is the corresponding combination of L and H?*

Exercise 10.22. *Is there a transducer Q which recognises the relation of all pairs $(0^n, 1^n 2^n)$ with $n \in \mathbb{N}$?*

Exercise 10.23. *Assume that the alphabet is $\Sigma = \{0, 1\}$. Construct a transducer R which accepts a triple (u, v, w) iff there is a common subsequence of length at least $|u|$ of v and w. Here x is a subsequence of y iff x can be split into parts $x_1, x_2, \ldots, x_k$ with $y \in \Sigma^* \cdot x_1 \cdot \Sigma^* \cdot x_2 \cdot \ldots \cdot \Sigma^* \cdot x_k \cdot \Sigma^*$.*

Exercise 10.24. *Is there a transducer S which recognises a pair (v, w) iff $v = w^{mi}$, that is, v is the mirror image of w?*

Exercise 10.25. *Is there a transducer T which recognises a pair (v, w) iff v occurs in w two times as a subword?*

Exercise 10.26. *Is there a transducer U which recognises all pairs (v, w) such that in v, w occur the same symbols the same number of times?*

Selftest 10.27. *Let f be an automatic function from a regular set A to a regular set B. Let $B_n = \{y \in B : y = f(x)$ for exactly n words $x \in A\}$ and let $B_\infty = \{y \in B : y = f(x)$ for infinitely many $x \in A\}$. Which of the sets $B_0, B_1, \ldots, B_\infty$ are regular?*

Selftest 10.28. *Assume that a group is generated by elements a, b, c and their inverses $\overline{a}, \overline{b}, \overline{c}$ and has the following rules: $a \circ b = c \circ a$, $a \circ c = b \circ a$, $b \circ c = c \circ b$ and correspondingly for the inverses, that is, $a \circ \overline{b} = \overline{c} \circ a$, $\overline{a} \circ b = c \circ \overline{a}$ and so on.*

Show that this group is fully automatic by providing a fully automatic representation and explain why the group operation $conv(x, y) \mapsto x \circ y$ is an automatic function (with two inputs) in this representation.

Selftest 10.29. *Let $L = \{00, 11\}^*$ be a regular language over the alphabet $\{0, 1, 2, 3\}$. Determine the syntactic monoid G_L for this language.*

Selftest 10.30. *Let $G = \{a^* \cup \overline{a}^*\} \cdot \{b^* \cup \overline{b}^*\}$ be a representation for the automatic group generated by a, b and the inverses $\overline{a}, \overline{b}$ with the rule $a \circ b = b \circ a$. Let L be the set of all strings over $\{a, \overline{a}, b, \overline{b}\}$ which are equivalent to the neutral element ε in this group.*

What is the complexity of L? (a) regular, (b) context-free and not regular, (c) context-sensitive and not context-free, (d) recursively enumerable and not context-sensitive.

Give a justification for the taken choice.

Selftest 10.31. *Let $\{L_d : d \in I\}$ and $\{H_e : e \in J\}$ be two automatic families with regular sets of indices I, J. Prove or disprove the following claim.*

Claim. There are an automatic relation $R \subseteq I \times I$ and an automatic function $f : R \to J$ with domain R such that for all $d, d' \in I$ the following statement holds: if there is an $e \in J$ with $L_d \cap L_{d'} = H_e$ then $(d, d') \in R$ and $H_{f(d,d')} = H_e$ else $(d, d') \notin R$.

Selftest 10.32. *Is every function which is computed by a nondeterministic Moore machine also computed by a deterministic Moore machine?*

If the answer is "yes" then explain how the Moore machine is made deterministic; if the answer is "no" then give an example of a function which is computed only by a nondeterministic Moore machine and not by a deterministic one.

Selftest 10.33. *Construct a Mealy machine which recognises the following function: $f(x)$ doubles every 0 and triples every 1 if x does not contain a 2; $f(x)$ omits all 0 and 1 from the input if x contains a 2.*

Sample outputs of f are $f(01) = 00111$, $f(01001) = 001110000111$, $f(021) = 2$ and $f(012210012) = 222$.

Solution for Selftest 10.27. All of the sets $B_0, B_1, \ldots, B_\infty$ are regular. The reason is that one can first-order define each of the sets using automatic functions (like f) and relations (like membership in A and length-lexicographic order). For example,

$$y \in B_0 \Leftrightarrow y \in B \wedge \forall x \in A\,[y \neq f(x)]$$

and

$$\begin{aligned} y \in B_2 \quad \Leftrightarrow \quad & \exists x_1, x_2 \in A\, \forall x \in A \\ & [f(x_1) = y \wedge f(x_2) = y \wedge x_1 \neq x_2 \wedge f(x) = y \rightarrow (y = x_1 \vee y = x_2)]. \end{aligned}$$

The formula for B_∞ has to be a bit different, as one has to say that there are infinitely many x which are mapped to y. For this one assumes that A is infinite, as otherwise $B_\infty = \emptyset$. Now the formula is

$$y \in B_\infty \Leftrightarrow \forall x \in A\, \exists x' \in A\,[x <_{ll} x' \wedge f(x') = y].$$

It is okay to introduce new automatic parameters (like the length-lexicographic ordering on A) in order to show that some set or relation or function is regular / automatic by providing the corresponding first-order definition.

Solution for Selftest 10.28. Take any fully automatic representation $(A, +)$ of the integers and note that the set $B = \{x : \exists y\,[y + y = x]\}$ of the even integers is regular. Now represent the group $(G, \circ)$ by $\{conv(i, j, k) : i, j, k \in A\}$ where $conv(i, j, k)$ stands for $a^i \circ b^j \circ c^k$. Now $conv(i, j, k) \circ conv(i', j', k')$ is $conv(i + i', j + j', k + k')$ in the case that i' is even and $conv(i + i', k + j', j + k')$ in the case that i' is odd. As B is regular, this case-distinction is automatic; furthermore, the addition is automatic in A and can therefore be carried out on the components.

Solution for Selftest 10.29. For the language $\{00, 11\}^*$, one has first to make the minimal dfa which has four states, namely s, z, o, t. Its transition table is the following (where s is the starting state):

state	acc/rej	0	1	2	3
s	accept	z	o	t	t
z	reject	s	t	t	t
o	reject	t	s	t	t
t	reject	t	t	t	t

The corresponding monoid has the following function f_u where for each f_u only one representative word u is taken.

word u	$f_u(s)$	$f_u(z)$	$f_u(o)$	$f_u(t)$
ε	s	z	o	t
0	z	s	t	t
00	s	z	t	t
001	o	t	t	t
0011	s	t	t	t
01	t	o	t	t
011	t	s	t	t
0110	t	z	t	t
1	o	t	s	t
10	t	t	z	t
100	t	t	s	t
1001	t	t	o	t
11	s	t	o	t
110	z	t	t	t
2	t	t	t	t

Solution for Selftest 10.30. The answer should be (c) "context-sensitive and not context-free". Let L_a be the set of all words in $\{a, \overline{a}, b, \overline{b}\}^*$ which have as many a as $\overline{a}$ and L_b be the set of all words in $\{a, \overline{a}, b, \overline{b}\}^*$ which have as many b as $\overline{b}$. Then $L = L_a \cap L_b$, thus L is the intersection of two context-free languages and therefore context-sensitive or context-free or regular.

Note that all levels of the Chomsky hierarchy are closed with respect to intersection with regular sets. Now one forms the set $L \cap a^*(\overline{b}\overline{a})^*b^*$. This set consists of all words of the form $a^n(\overline{b}\overline{a})^n b^n$ and this is a well-known example of a context-sensitive language which is not context-free. Therefore the language L cannot be regular and cannot be context-free; so context-sensitive is the right level.

Solution for Selftest 10.31. The claim is true. The main idea is that the function f plus its domain is first-order definable from automatic parameters. Indeed, one can introduce a relation R' and then derive R, f from R' as follows:

$$\begin{aligned}(d, d', e) \in R' \quad &\Leftrightarrow \quad d, d' \in I \wedge e \in J \wedge \forall x\, [x \in H_e \Leftrightarrow x \in L_d \wedge x \in L_{d'}];\\ (d, d') \in R \quad &\Leftrightarrow \quad \exists e \in J\, [R'(d, d', e)];\\ f(d, d') = e \quad &\Leftrightarrow \quad R'(d, d', e) \wedge \forall e' \in J\, [R'(d, d', e') \Rightarrow e \leq_{ll} e'].\end{aligned}$$

Here again one uses the automatic length-lexicographic ordering and the automaticity of the membership problem of the corresponding automatic families.

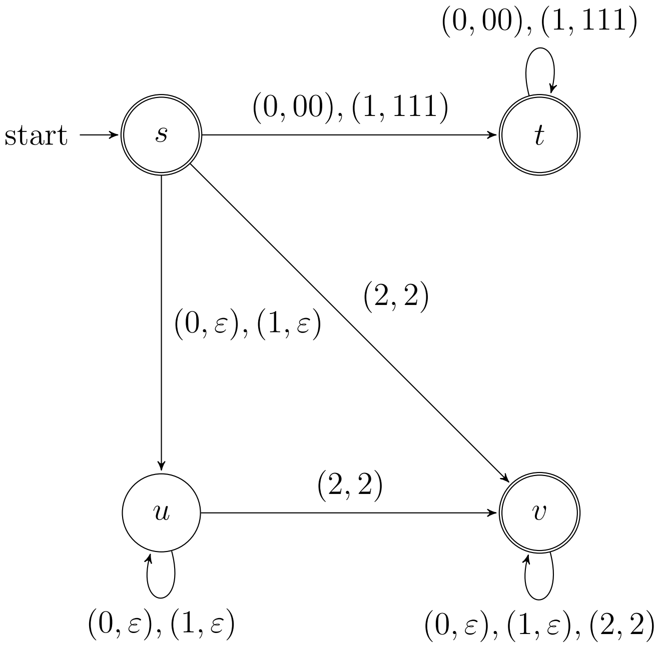
Solution for Selftest 10.32. The answer is "no" and the reason is that a nondeterministic Moore machine can anticipate some information which the deterministic

Moore machine cannot anticipate.

For example, a Moore machine should map any input $a_0 a_1 \ldots a_n$ to $(a_n)^{n+1}$, that is, replace all a_m by the last digit a_n. For this the Moore machine needs nondeterministically to anticipate what a_n is. So the Moore machine has a start state s without any output and for each symbol $a \in \Sigma$ it has two states r_a (rejecting) and q_a (accepting). Now on the first symbol b the Moore machine nondeterministically chooses a and if $b = a$ then it goes to q_a else it goes to r_a. On each further symbol c, if $a = c$ then the machine goes to q_a else it goes to r_a. Both states r_a and q_a output in each cycle one symbol a. If the last input symbol is a then the automaton will be in the accepting state q_a else it will be in the rejecting state r_a. So if the input ends with a the run is accepting and the output is correct; if the input does not end with a then the run ends in a rejecting state and the output is not valid.

A deterministic Moore machine cannot compute this function. If the Moore machine sees an input 0, it needs to respond with a 0 immediately, as it otherwise would not map 0 to 0, hence it goes on 0 to a state with output 0. If then a 1 follows, the output has to be 11, what is impossible for the deterministic Moore machine to do, as it has already written a 0.

Solution for Selftest 10.33.



11 Regular Languages and Learning Theory

Angluin [3] investigated the question on how to learn a dfa by a dialogue between a learner (pupil) and teacher. The learner can ask questions to the teacher about the concept (dfa) to be learnt and the teacher answers. The learner can ask two types of questions:

- Is the following dfa equivalent to the one to be learnt?
- Does the dfa to be learnt accept or reject the following word w?

The first type of questions are called "equivalence queries" and the second type of questions are called "membership queries". The teacher answers an equivalence query either with "YES" (then the learner has reached the goal) or "NO" plus a counterexample w on which the dfa given by the learner and the dfa to be learnt have different behaviour; the teacher answers a membership query by either "YES" or "NO".

Theoretically, the learner could just take a listing of all dfas and ask "Is dfa$_1$ correct?", "Is dfa$_2$ correct?", "Is dfa$_3$ correct?" ... and would need s equivalence queries to find out whether dfa$_s$ is correct. This strategy is, however, very slow; as there are more than 2^n dfas with n states, one would for some dfas with n states need more than 2^n queries until the dfa is learnt. Angluin showed that there is a much better algorithm and she obtained the following result.

Theorem 11.1: Angluin's algorithm to learn dfas by queries [3]. *There is a learning algorithm which has polynomial response time in each step and which learns in time polynomial in the number of states of the dfa to be learnt and the longest counterexample given an arbitrary dfa using equivalence queries and membership queries.*

Proof. A simplified version of Angluin's algorithm is given. The idea of Angluin is that the learner maintains a table (S, E, T) which is updated in each round. In this table, S is a set of words which represent the set of states. E consists of all the counterexamples observed plus their suffixes. S and E are finite sets of size polynomial in the number and length of counterexamples seen so far and T is a function which for all members $w \in S \cdot E \cup S \cdot \Sigma \cdot E$ says whether the automaton to be learnt accepts or rejects w.

Angluin defines the notion of a row: For $u \in S \cup S \cdot \Sigma$, let $(v_1, v_2, \ldots, v_k)$ be a listing of the current elements in E and for each $u \in S \cup S \cdot \Sigma$, let the vector $row(u)$ be $(T(uv_1), T(uv_2), \ldots, T(uv_k))$. The table (S, E, T) is called closed, if for every $u \in S$ and $a \in \Sigma$ there is a $u' \in S$ with $row(u') = row(ua)$.

Now Anlguin defines for each closed (S, E, T) the finite automaton DFA(S, E, T) where the set of states is S, the alphabet is Σ and the transition function finds to a state u and $a \in \Sigma$ the unique $u' \in S$ with $row(u') = row(ua)$. The starting state is represented by the empty word ε (which is in S). A state u is accepting iff $T(u) = 1$. Note that DFA(S, E, T) is complete and deterministic. The learning algorithm is now the following.

```
Teacher has regular set $L$ and learner makes membership and equivalence queries.
1. Initialise $S = \{\varepsilon\}$ and $E = \{\varepsilon\}$.
2. For all $w \in S \cdot E \cup S \cdot \Sigma \cdot E$ where $T(w)$ is not yet defined, make a
   membership query to determine $L(w)$ and let $T(w) = L(w)$.
3. If there are $u \in S$ and $a \in \Sigma$ with $row(ua) \neq row(u')$ for all $u' \in S$
   then let $S = S \cup \{ua\}$ and go to 2.
4. Make an equivalence query whether DFA$(S, E, T)$ recognises $L$.
5. If the answer is "YES" then terminate with DFA$(S, E, T)$.
6. If the answer is "NO" with counterexample $w$ then let $E = E \cup \{v :
   \exists u\, [uv = w]\}$ and go to 2.
```

Now one shows various properties in order to verify the termination of the algorithm and the polynomial bounds on the number of membership and equivalence queries. For this, assume that $(Q, \Sigma, \delta, s, F)$ is the minimal dfa recognising L. Now various invariants are shown.

If u, u' are different elements of S then $\delta(s, u) \neq \delta(s, u')$ as there is a word $v \in E$ with $L(uv) \neq L(u'v)$. Hence $|S| \leq |Q|$ throughout the algorithm.

If (S, E, T) is closed and $w \in E$ then the DFA accepts w iff $w \in L$.

To see this, one does the following induction: Let $w = a_1 a_2 \ldots a_n$. Clearly $T(w) = L(w)$ by the corresponding membership query. For $m = 0, 1, \ldots, n$, one shows that the automaton is after processing $a_1 a_2 \ldots a_m$ is in a state u_m with $T(u_m a_{m+1} a_{m+2} \ldots a_n) = T(w)$. This is true for $m = 0$ as $u_0 = \varepsilon$ is the initial state of DFA(S, E, T). Assume now that it is correct for $m < n$. Then u_{m+1} is the unique state in S with $row(u_{m+1}) = row(u_m a_{m+1})$. It follows that $T(u_m a_{m+1} v) = T(u_{m+1} v)$ for $v = a_{m+2} a_{m+3} \ldots a_n$. Hence the induction hypothesis is preserved and DFA(S, E, T) is after processing the full word in a state u_n with $T(u_n) = T(w)$. This state is accepting iff $T(u_n) = 1$ iff $T(w) = 1$. Hence DFA(S, E, T) is correct on w.

Assume that the algorithm has the parameters (S, E, T) before observing counterexample w and has the parameters (S', E', T') after it has done all the updates before the next equivalence query is made. Then $S \subset S'$.

Let $row_E(u)$ and $row_{E'}(u)$ denote the rows of u based on E and E'; note that $E \subseteq E'$ and therefore $row_E(u) \neq row_E(u') \Rightarrow row_{E'}(u) \neq row_{E'}(u')$. Now, as DFA$(S, E, T) \neq$ DFA(S', E', T') on w, the states in which these two automata are after processing w must be different. As both dfas have the initial state ε, there must be a first prefix of the form ua of w such that the two automata are in different states u', u'' after processing ua. Now $row_E(ua) = row_E(u')$ and $u' \in S$ and $row_{E'}(ua) = row_{E'}(u'')$. It cannot be that $u'' \in S - \{u'\}$, as then $row_E(u'') \neq row_E(ua)$. Hence u'' must be a new state in $S' - S$ and $S \subset S'$.

Let r be the sum of all lengths of the counterexamples observed. The algorithm makes at most $|Q|$ equivalence queries and at most $|Q| \cdot (|\Sigma|+1) \cdot (r+1)$ membership queries.

As seen, $|S| \leq |Q|$ throughout the algorithm. As each equivalence query increases the size of S, there are at most $|Q|$ equivalence queries. Furthermore, E contains all non-empty prefixes of counterexamples observed plus ε, hence $|E| \leq r+1$. Now the table T has at each time the domain $S \cdot E \cup S \cdot \Sigma \cdot E$ what gives then the bound on the number of membership queries.

The overall runtime of each update is polynomial in the size of the counterexamples observed so far and in $|Q|$. So latest when $|S| = |Q|$ the answer to the equivalence query is "YES" and the learner has learnt the language L. ∎

Remark 11.2: Angluin's original algorithm [3]**.** Angluin did not put the suffixes of the counterexamples into E but she put the prefixes of the counterexamples into S. Therefore, S could contain words u, u' with $row(u) = row(u')$. In order to avoid that this is harmful, Angluin increased then E so long until the table T is consistent, that is, if $row(u) = row(u')$ then $row(ua) = row(u'a)$ for all $u, u' \in S$ and $a \in \Sigma$. This consistency requirement was explicitly added into the algorithm. The verification of the original algorithm is given in Angluin's paper [3].

```
Teacher has regular set $L$ and learner makes membership and equivalence
    queries.
1. Initialise $S = \{\varepsilon\}$ and $E = \{\varepsilon\}$.
2. For all $w \in S \cdot E \cup S \cdot \Sigma \cdot E$ where $T(w)$ is not yet defined, make a
    membership query to determine $L(w)$ and let $T(w) = L(w)$.
3. If there are $u, u' \in S$, $a \in \Sigma$ and $v \in E$ such that $row(u) = row(u')$
    and $T(uav) \neq T(u'av)$ then let $E = E \cup \{av\}$ and go to 2.
4. If there are $u \in S$ and $a \in \Sigma$ with $row(ua) \neq row(u')$ for all $u' \in S$
    then let $S = S \cup \{ua\}$ and go to 2.
5. Make an equivalence query whether DFA($S, E, T$) recognises $L$.
6. If the answer is "YES" then terminate with DFA($S, E, T$).
7. If the answer is "NO" with counterexample $w$ then let $S = S \cup \{u :$
    $\exists v\,[uv = w]\}$ and go to 2.
```

Description 11.3: Learning from positive data [2, 4, 36]**.** Gold [36] introduced a general framework of learning in the limit. His idea was that a learner reads more and more data and at the same time outputs conjectures; from some time on, the learner should always output the same correct conjecture. More precisely, the learner consists of a memory mem and an update function uf. In each round, the update function uf maps pairs (mem, x) consisting of the current memory and a current datum x observed to pairs (mem', e) where mem' is the new memory which is based on some calculations and intended to have incorporated some way to memorise x and where e is the conjectured hypothesis. In the case of learning regular languages, this hypothesis could just be a dfa. Gold [36] observed already in his initial paper that a class is unlearnable iff it contains an infinite set and all of its finite sets. As Σ^* and each of its finite subsets is regular, the class of regular sets is not learnable from positive data. Nevertheless, one still might learn some subclasses of regular languages.

For this, one considers so called automatic families. An automatic family is given by an index set I and a family of sets $\{L_d : d \in I\}$ such that the relation of all (d, x) with $x \in L_d$ is automatic, that is, the set $\{conv(d, x) : d \in I \wedge x \in L_d\}$ is a regular set.

Here the size of the minimal index of each language is invariant up to a constant with respect to different indexings. So given two indexed families $\{L_d : d \in I\}$ and $\{H_e : e \in J\}$, one can define the automatic functions $i : I \to J$ and $j : J \to I$ with $i(d) = \min_{ll}\{e \in J : H_e = L_d\}$ and $j(e) = \min_{ll}\{d \in I : L_d = H_e\}$. Then there is a constant k such that the following holds: if $i(d)$ is defined then $|i(d)| \leq |d| + k$; if $j(e)$ is defined then $|j(e)| \leq |e| + k$. Hence the sizes of the minimal indices of a language in both families differ at most by k [48].

The data on the language L to be learnt are presented in form of a text. A text T for a language L is an infinite sequence of words and pause symbols # such that $L = \{w : w \neq \# \wedge \exists n\, [T(n) = w]\}$. The learner starts now with some fixed initial memory mem_0 and initial hypothesis e_0, say ε and a hypothesis for the empty set. In round n, the new memory and hypothesis are computed by the update function uf: $(mem_{n+1}, e_{n+1}) = uf(mem_n, T(n))$. The learner learns L using the hypothesis space $\{L_d : d \in I\}$ iff there is a $d \in I$ with $L_d = L$ and $\forall^\infty n\, [e_n = d]$.

Angluin [2] showed in a very general framework a learnability result which covers the case of automatic families.

Theorem 11.4: Angluin's tell-tale criterion [2]. *An automatic family $\{L_d : d \in I\}$ is learnable from positive data iff there is for every $d \in I$ a finite subset $F_d \subseteq L_d$ such that there is no further index e with $F_d \subseteq L_e \subset L_d$.*

Proof. Assume that $\{L_d : d \in I\}$ has a learner. Blum and Blum [4] showed that for each L_d there must be a finite initial part $T(0)T(1)\ldots T(n)$ of a text for L_d such that for every extension $T(n+1)T(n+2)\ldots T(m)$ using elements from L_d and pause symbols # it holds that the learner conjectures an index for L_d after processing $T(0)T(1)\ldots T(m)$. If such an initial part would not exist, one could inductively define a text T for L_d on which the learner infinitely often outputs an index for a set different from L_d. Now $F_d = \{T(m) : m \leq n \wedge T(m) \neq \#\}$. This is obviously a subset of L_d; furthermore, when seeing only data of L_d after this initial part $T(0)T(1)\ldots T(n)$, the learner outputs a conjecture for L_d, hence the learner does not learn any proper subset of L_d from a text starting with $T(0)T(1)\ldots T(n)$. It follows that there cannot be any e with $F_d \subseteq L_e \subset L_d$.

For the other direction, consider that the sets F_d exist. Therefore the following value $f(d)$ is defined for every $d \in I$:

$$f(d) = \min{}_{ll}\{b : \forall e \in I\ \ [\{x \in L_d : x \leq_{ll} b\} \subseteq L_e \subseteq L_d \Rightarrow L_e = L_d]\}.$$

Then it is clear that one can choose $F_d = \{x \in L_d : x \leq_{ll} f(b)\}$. One can first-order define the subset-relation and equality-relation on sets:

$$\begin{aligned} L_d \subseteq L_e &\ \Leftrightarrow\ \forall x\,[x \in L_d \Rightarrow x \in L_e]; \\ L_d = L_e &\ \Leftrightarrow\ \forall x\,[x \in L_d \Leftrightarrow x \in L_e]. \end{aligned}$$

Hence the function f is first-order definable using automatic parameters and is automatic. Thus one can make the following learning algorithm which for doing its search archives all the data seen so far: mem_n is a list of data seen so far; e_n is the least member of I which satisfies that all elements of L_d up to $f(d)$ have been observed so far and no non-elements of L_d have been observed prior to round n; if such an index does currently not exist, the learner can output ? in order to signal that there is no valid hypothesis.

Assume that the learner reads a text for a language and that d is the minimal index of this language. Assume that n is so large that the following condition are satisfied:

- For every $w \in L_d$ with $w \leq_{ll} f(d)$ there is an $m < n$ with $T(m) = w$;
- For every $e <_{ll} d$ with $L_e \not\supseteq L_d$ there is an $m < n$ with $T(m) \in L_d - L_e$.

Note that if $e <_{ll} d$ and $L_e \supset L_d$ then there must be an element $w \in L_e - L_d$ with $w \leq_{ll} f(e)$; this element does not appear in the text T. Hence, for the n considered above it holds that the hypothesis of the learner is d. Thus the learner converges on the text T to the minimal index d of the language described by the text T; it follows that the learner learns the family $\{L_d : d \in I\}$ from positive data. ∎

This characterisation answers when a class is learnable in general. One could now ask what additional qualities could be enforced on the learner for various classes. In particular, can one make the update function *uf* automatic? Automatic learners are defined as follows.

Description 11.5: Automatic learners [12, 47, 48]**.** An automatic learner is given by its initial memory mem_0, initial hypothesis e_0 and the update function *uf* which computes in round n from $conv(mem_n, x_n)$ the new memory and the hypothesis, represented as $conv(mem_{n+1}, e_{n+1})$. An automatic learner for an indexed family $\{L_d : d \in I\}$ (which is assumed to be one-one) might use another hypothesis space $\{H_e : e \in J\}$ and must satisfy that there is an n with $H_{e_n} = L_d$ and $e_m \in \{e_n, ?\}$ for all $m > n$ where ? is a special symbol the learner may output if memory constraints do not permit the learner to remember the hypothesis.

Memory constraints are there to quantify the amount of information which an automatic learner is permitted to archive on data seen in the past. In general, this data never permits to recover the full sequence of data observed, although it is in many cases still helpful. The following memory constraints can be used while learning L_d where the current conjecture of the learner is $H_{e_{n+1}}$ and where $x_0, x_1, \ldots, x_n$ are the data observed so far; i is the function with $L_{i(e)} = H_e$ for all e representing a language in the class to be learnt.

None: The automaticity of *uf* gives that even in the absence of an explicit constraint it holds that $|mem_{n+1}| \leq \max\{|x_n|, |mem_n|\} + k$ for some constant k and all possible values of mem_n and x_n.

Word-sized: $|mem_{n+1}| \leq \max\{|x_0|, |x_1|, \ldots, |x_n|\} + k$ for some constant k.

Hypothesis-sized: $|mem_{n+1}| \leq |e_{n+1}| + k$ for some constant k.

Original-hypothesis-sized: $|mem_{n+1}| \leq |i(e_{n+1})| + k$ for some constant k with the additional constraint that $i(e_{n+1})$ is defined, that is, $H_{e_{n+1}}$ must be in the class to be learnt.

Target-sized: $|mem_{n+1}| \leq |d| + k$ for some constant k.

Constant: $mem_{n+1} \in C$ for a fixed finite set C of possible memory values.

Memoryless: $mem_{n+1} = mem_0$.

Note that target-sized always refers to the size of the original target; otherwise the constraint would not be the same as hypothesis-sized, as the learner could use a hypothesis space where every language has infinitely many indices and would choose at every revision a hypothesis longer than the current memory size. Note that one could fix the constant k to 1 for word-sized, hypothesis-sized, original-hypothesis-sized and target-sized learners as one can adjust the alphabet-size and store the last k symbols in a convolution of one symbol. As this would, however, make the construction of learners at various times more complicated, it is easier to keep the constant k unspecified.

Furthermore, a learner is called iterative iff $mem_n = e_n$ for all n and e_0 is a hypothesis for the empty set (which is added to the hypothesis space, if needed). Iterative learners automatically have a hypothesis-sized memory; furthermore, one writes $uf(e_n, x_n) = e_{n+1}$ in place of $uf(e_n, x_n) = conv(e_{n+1}, e_{n+1})$ in order to simplify the notation.

Example 11.6. If I is finite then there is a bound b such that for all different d, d' there is an $w \leq_{ll} b$ which is in one but not both of $L_d, L_{d'}$. Hence one can make a learner which memorises for every $w \leq_{ll} b$ whether the datum w has been observed. In the limit, the learner knows for every $w \leq_{ll} b$ whether $w \in L_d$ for the language L_d to be learnt and therefore the learner will eventually converge to the right hypothesis. The given learner has constant-sized memory.

If one would require that the learner repeats the correct conjecture forever once it has converged to the right index, then only finite classes can be learnt with constant-sized memory. If one permits ? after convergence, then a memoryless learner can learn the class of all $L_d = \{d\}$ with $d \in I$ for any given infinite regular I: the learner outputs d on datum d and ? on datum # and does not keep any records on the past.

Example 11.7. Assume that $\Sigma = \{0, 1, 2\}$ and that $I = \{conv(v, w) : v, w \in \Sigma^* \wedge v \leq_{lex} w\} \cup \{conv(3, 3)\}$ with $L_{conv(v,w)} = \{u \in \Sigma^* : v \leq_{lex} u \leq_{lex} w\}$ for all $conv(v, w) \in I$. Note that $L_{conv(3,3)} = \emptyset$.

This class has an iterative learner whose initial memory is $conv(3, 3)$. Once it sees a word $u \in \Sigma^*$, the learner updates to $conv(u, u)$. From that onwards, the learner updates the memory $conv(v, w)$ on any word $u \in \Sigma^*$ to $conv(\min_{lex}\{u, v\}, \max_{lex}\{u, w\})$. This hypothesis always consists of the convolution of the lexicographically least and greatest datum seen so far and the sequence of hypotheses has converged once the learner has seen the lexicographically least and greatest elements of the set to be learnt (which exist in all languages in the class to be learnt).

Data seen so far	Hypothesis	Language of hypothesis
—	conv(3,3)	$\emptyset$
#	conv(3,3)	$\emptyset$
# 00	conv(00,00)	$\{00\}$
# 00 0000	conv(00,0000)	$\{00, 000, 0000\}$
# 00 0000 1	conv(00,1)	$\{u : 00 \leq_{lex} u \leq_{lex} 1\}$
# 00 0000 1 0	conv(0,1)	$\{u : 0 \leq_{lex} u \leq_{lex} 1\}$
# 00 0000 1 0 112	conv(0,112)	$\{u : 0 \leq_{lex} u \leq_{lex} 112\}$
# 00 0000 1 0 112 011	conv(0,112)	$\{u : 0 \leq_{lex} u \leq_{lex} 112\}$

Exercise 11.8. *Make an automatic learner which learns the class of all $L_d = \{dw : w \in \Sigma^*\}$ with $d \in \Sigma^*$; that is, $I = \Sigma^*$ in this case.*

Exercise 11.9. *Assume that a class $\{L_d : d \in I\}$ is given with $L_d \neq L_{d'}$ whenever $d, d' \in I$ are different. Assume that an automatic learner uses this class as a hypothesis space for learning satisfying any of the constraints given in Description 11.5. Let $\{H_e : e \in J\}$ be any other automatic family containing $\{L_d : d \in I\}$ as a subclass. Show that there is an automatic learner satisfying the same type of memory constraints conjecturing indices taken from J in place of I.*

Theorem 11.10: Jain, Luo and Stephan [47]. *Let $I = \Sigma^*$, $L_\varepsilon = \Sigma^+$ and $L_d = \{w \in \Sigma^* : w <_{ll} d\}$ for $d \in \Sigma^+$. The class $\{L_d : d \in I\}$ can be learnt using a word-sized memory but not using an hypothesis-sized memory.*

Proof. First assume by way of contradiction that a learner could learn the class using some chosen hypothesis space with hypothesis-sized memory. Let $T(n)$ be the n-th string of Σ^+. When learning from this text, the learner satisfies $e_m = e_{m+1}$ for some n and all $m \geq n$; furthermore, $H_{e_m} = \Sigma^+$ for these $m \geq n$. Therefore, from n onwards, all values of the memory are finite strings of length up to $|e_n| + k$ for some constant k. There are only finitely many such strings and therefore there must be $m, k \geq n$ with $mem_m = mem_k$. If one now would change the text to $T(h) = \varepsilon$ for all $h \geq m$ or $h \geq k$, respectively, the learner would converge to the same hypothesis on both of these texts, although it would be a text for either the first $m + 1$ or the first $k + 1$ strings in Σ^*. Thus the learner fails to learn at least one of these finite sets and cannot learn the class.

Second consider a word-sized learner. This learner memorises the convolution of the length-lexicographically least and greatest words seen so far. There are three cases:

- In the case that no word has been seen so far, the learner outputs ? in order to abstain from a conjecture;
- In the case that these words are ε and v, the learner conjectures $L_{Succ_{ll}(v)} = \{w : \varepsilon \leq_{ll} w \leq_{ll} v\}$;
- In the case that the words u and v memorised are different from ε, the learner conjectures $L_\varepsilon = \Sigma^+$.

The memory of the learner is either a special symbol for denoting that no word (except #) has been seen so far or the convolution of two words observed whose length is bounded by the length of the longest word seen so far. Hence the memory bound of the learner is satisfied. ▮

Theorem 11.11. *Assume that $\Sigma = \{0,1\}$ and $I = \{0,1\}^* \cup \{2,3\} \cup \{conv(v,w) : v,w \in \{0,1\}^* \wedge v <_{ll} w\}$ where the convolution is defined such that this unions are disjoint. Furthermore, let $L_2 = \emptyset$, $L_3 = \Sigma^*$, $L_v = \{v\}$ for $v \in \Sigma^*$ and $L_{conv(v,w)} = \{v,w\}$ for $v,w \in \Sigma^*$ with $v <_{ll} w$. The class $\{L_d : d \in I\}$ can neither be learnt with constant memory nor with target-sized memory. It can, however, be learnt using an original-hypothesis-sized memory.*

Proof. Assume that some learner with constant-sized memory learns this class. There is a constant k so large that (1) $|e_n| \leq |x_n|+k$ on datum x_n and at memory $mem_n \in C$ and (2) $|\max(H_e)| \leq |e|+k$ whenever H_e is finite. As C has only finitely many values, this constant k must exist. Now assume that v is any member of Σ^* and $w \in \Sigma^*$ is such that $|w| > |v|+2k+1$. Then, whenever the hypothesis e_{m+1} is computed from e_m and either v or #, the set $H_{e_{m+1}}$ is neither $\{w\}$ nor $\{v,w\}$. Hence, when the learner sees w as the first datum, it must conjecture $\{w\}$ as all subsequent data might by # and $\{w\}$ cannot be conjectured again. Furthermore, if the learner subsequently sees only v, then it cannot conjecture $\{v,w\}$. Hence, either the learner does not learn $\{w\}$ from the text $w, \#, \#, \ldots$ or the learner does not learn $\{v,w\}$ from the text $w, v, v, \ldots$; thus the learner does not learn the given class.

As Σ^* is in the class to be learnt and every data observed is consistent with the possibility that Σ^* is the language observed, every target-sized learner has at every moment to keep the index shorter than the index of Σ^* plus some constant, hence this learner has actually to use constant-sized memory what is impossible by the previous paragraph.

So it remains to show that one can learn the class by hypothesis-sized memory. This is done by showing that the class has actually an iterative learner using I as hypothesis space. Hence every hypothesis is from the original space and so the learner's memory is original-hypothesis-sized. Initially, the learner conjectures 2 until it sees a datum $v \neq \#$. Then it changes to conjecturing H_v until it sees a datum $w \notin \{\#, v\}$. Then the learner updates to $H_{conv(\min_{ll}\{v,w\},\max_{ll}\{v,w\})}$. The learner keeps this hypothesis until it sees a datum outside $\{\#, v, w\}$; in that case it makes a last mind change to $H_3 = \Sigma^*$. It is easy to see that the learner is iterative and needs only the current hypothesis as memory; furthermore, the learner is also easily seen to be correct. ▮

Theorem 11.12. *If a learner learns a class with target-sized memory then the learner's memory is also word-sized on texts for languages in the class.*

Proof. Let a learner with target-sized memory be given and k be the corresponding constant. Whenever the learner learns has seen examples $x_0, x_1, \ldots, x_n$ when $|mem_{n+1}| \leq |d| + k$ for all languages L_d which contain the data observed so far. Let x_m be the longest datum seen so far. Let e be the length-lexicographically first index with $\{x_0, x_1, \ldots, x_n\} \subseteq L_e \cup \{\#\}$. If e is shorter than some of the data then

$|mem_{n+1}| \leq |x_m| + k$. Otherwise let e' be the prefix of e of length $|x_m|$.

Consider the dfa which recognises the set $\{conv(d, x) : x \in L_d\}$. Let C be the set of those states which the automata takes on any $u \in L_e$ with $|u| \leq |e'|$ after having processed $conv(e', u)$; it is clear that the automaton will accept $conv(e, u)$ iff it is in a state in C after processing $conv(e', u)$. Hence one can define an automatic function f_C such that $f_C(d')$ is the length-lexicographically least index $d \in I$ such that

$$\forall u \in \Sigma^* \text{ with } |u| \leq |d'|$$
$$[u \in L_d \Leftrightarrow \text{the dfa has after processing } conv(d', u) \text{ a state in the set } C]$$

Now $f_C(d') \leq |d'| + k_C$ for some constant k_C and all d' where $f_C(d')$ is defined. Let k' be the maximum of all $k_{C'}$ where C' ranges over sets of states of the dfa. Furthermore, as $f_C(e')$ is defined and equal to e, one gets that $|e| \leq |f_C(e')| \leq |e'| + k' = |x_m| + k'$ and $|mem_{n+1}| \leq |e| + k \leq |x_m| + k + k'$. The constant $k + k'$ is independent of the language to be learnt and the text selected to present the data; hence the learner has word-sized memory on all texts belonging to languages in the class. ▮

Remark 11.13. The result can be strengthed by saying whenever a class is learnable with target-size memory then it is also learnable with word-size memory. Here the strengthening is that the learner keeps the memory bound also on texts which are for languages outside the class to be learnt.

For this, given an original learner having the word-size memory bound only on languages in the class (with a constant k), one can make a new learner which either has as memory $conv(mem_n, x_m)$ where mem_n is the memory of the original learner and x_m is the longest word seen so far or it has a special value ?. The initial memory is $conv(mem_0, \#)$ and on word x_n it is updated from $conv(mem_n, x_m)$ according to that case which applies:

1. $conv(mem_{n+1}, x_m)$ if $|x_n| < |x_m|$ and $|mem_{n+1}| \leq |x_m| + k$;
2. $conv(mem_{n+1}, x_n)$ if $|x_n| \geq |x_m|$ and $|mem_{n+1}| \leq |x_n| + k$;
3. ? if $|mem_{n+1}| > \max\{|x_m|, |x_n|\} + k$.

Here mem_{n+1} is the memory computed from mem_n and x_n according to the original learner. The hypothesis e_{n+1} of the original learner is taken over in the case that the new memory is not ? and the hypothesis is ? in the case that the new memory is also ?. Note that the special case of the memory and hypothesis being ? only occurs if the original learner violates the word-size memory constraint and that only occurs in the case that the text $x_0, x_1, x_2, \ldots$ is not for a language in the class to be learnt.

Exercise 11.14. *Assume that $\{L_d : d \in I\}$ is the class to be learnt and that every language in the class is finite and that for every language in the class there is exactly* *one index in I. Show that if there is a learner using word-sized memory for this class, then the memory of the same learner is also target-sized. For this, show that there is a constant k such that all $d \in I$ and $x \in L_d$ satisfy $|x| \leq |d| + k$ and then deduce the full result.*

Exercise 11.15. *Show that there is an automatic family $\{L_d : d \in I\}$ such that I contains for each L_d exactly one index and the L_d are exactly the finite subsets of $\{0\}^*$ with even cardinality. Show that the class $\{L_d : d \in I\}$ has an iterative learner using the given hypothesis space. Is the same possible when the class consists of all subsets of $\{0\}^*$ with 0 or 3 or 4 elements? Note that an iterative learner which just conjectured an $d \in I$ must abstain from updating the hypothesis on any datum $x \in L_d$.*

Exercise 11.16. *Is the family of all finite subsets of $\{0\}^* \cdot \{1\}^*$ an automatic family? If so, then provide the corresponding index set and coding else explain why it cannot be automatic.*

Exercise 11.17. *Is the family of an infinite regular set L and all subsets of up to 5 elements an automatic family? If so, then provide the corresponding index set and coding else explain why it cannot be automatic.*

Exercise 11.18. *Is the family of all sets of decimal numbers which contain, for some $n > 0$, exactly two digits each n times and all other digits 0 times, an automatic family? If so, then provide the corresponding index set and coding else explain why it cannot be automatic.*

Exercise 11.19. *Consider an automatic family $\{L_e : e \in I\}$ such that for each two distinct $d, e \in I$ either $L_d \subset L_e$ or $L_e \subset L_d$ holds. Furthermore, assume that for each e there is a unique x_e such that $x_e \in L_e$ but $x_e \notin L_d$ for any d with $L_d \subset L_e$. Prove that the mapping $e \mapsto x_e$ is automatic and provide an automatic learner for the family.*

Exercise 11.20. *Given $\{L_d : d \in I\}$ and $\{L_e : e \in J\}$ both satisfying the conditions of Exercise 11.19, construct an automatic learner for the automatic family of all $K_{conv(d,e)}$ with $d \in I$, $e \in J$ and $K_{conv(d,e)} = \{0x : x \in L_d\} \cup \{1y : y \in H_e\}$.*

Exercise 11.21. *Consider the classes*

$$\begin{array}{lll}
\{L_e : e \in \{0\}^* & \text{with} & L_e = \{x \in \{0\}^* : |e| \leq |x|\}; \\
\{H_e : e \in \{0\}^* & \text{with} & H_e = \{x \in \{0\}^* : |e| \neq |x|\}; \\
\{K_e : e \in \{0\}^* & \text{with} & K_e = \{x \in \{0\}^* : |e| \geq |x|\}.
\end{array}$$

Which of these classes can be learnt with target-sized memory by an automatic learner? Provide the corresponding automatic learners or write why they do not exist.

Exercise 11.22. *Which of the classes in Exercise 11.21 can be learnt with hypothesis-sized memory? Provide the corresponding automatic learners or write why they do not exist.*

Exercise 11.23. *Which of the classes in Exercise 11.21 can be learnt with word-sized memory? Provide the corresponding automatic learners or write why they do not exist.*

Exercise 11.24. *Provide an infinite class learnable with constant memory size but not without any memory. Note that learners do not need to output the correct hypothesis all the time, but can also intermediately output* ?, *provided that there is a time where they output a correct hypothesis and that they do not output any other hypothesis (except* ?*) afterwards.*

12 Open Problems in Automata Theory

This chapter gives an overview of open problems in automata theory. First some problems left open from research here in Singapore are given, afterwards more difficult, generally open questions are presented.

First: Open Problems from Work in Singapore. *There are various open questions related to the memory-usage of automatic learners. These questions have not been solved in the past four years of research on automatic learning. The learners below are always understood to be automatic.*

Open Problem 12.1.

1. *Does every automatic family which has an automatic learner also have a learner with word-sized memory?*

2. *Does every automatic family which has a learner with hypothesis-sized memory also have a learner with word-sized memory?*

3. *Does every automatic family which has a learner with hypothesis-sized memory also have an iterative learner?*

In recursion-theory and complexity theory, one often looks at reducibilities which compare sets with functions, for example one has relations like

$$A \leq_m B \Leftrightarrow \exists f \, \forall x \, [A(x) = B(f(x))]$$

where the possible f are taken from a specific class. One could for example do the same with automatic functions. These notions can be refined, for example one can additionally ask that f has to be one-one. Then there are quite trivial examples of incomparable sets: When one fixes the alphabet $\{0,1\}$ then $\{0\}^*$ and $0^*1 \cdot \{0,1\}^*$ are incomparable, as either an exponential set has to be one-one mapped into a linear-sized one or the exponential complement of a set has to be one-one mapped into the linear-sized complement of another set. Both cannot be done, as the image of an exponentially growing set under an automatic function is again exponentially growing. For this, recall that a set A is linear-sized iff there is a linear function f such that A has at most $f(n)$ elements shorter than n; similarly, one can define when A is polynomial-sized and exponential-sized. Wai Yean Tan [85] worked with a slightly modified version where he ignores the alphabet and defines the notions just restricted to the sets to be compared.

Definition 12.2. *Let $A \leq_{au} B$ denote that there is an automatic function f such that*

$$\forall x, y \in A\, [f(x) \neq f(y) \wedge f(x) \in B].$$

Similarly one writes $A \leq_{tr} B$ for the corresponding definition where f is any function computed by a finite transducer.

Wai Yean Tan [85] investigated both notions. For his findings, one needs the following notions: A set A has size $\Theta(n^k)$ iff there a constant c such that up to length n there are at least $n^k/c - c$ and at most $n^k \cdot c + c$ elements in A. A regular set is polynomial-sized in the case that it has size $\Theta(n^k)$ for some k; a regular set is exponential-sized in the case that there is a constant c such that A has at least $2^{n/c} - c$ elements up to length n for each n. Note that every regular set is either finite or polynomial-sized or exponential-sized.

Theorem 12.3. *Let A, B be regular sets.*

1. *The sets A, B are comparable for tr-reducibility: $A \leq_{tr} B$ or $B \leq_{tr} A$. Furthermore, $A \leq_{tr} B$ if one of the following conditions holds:*
   - *A, B are both finite and $|A| \leq |B|$;*
   - *A is finite and B infinite;*
   - *A has size $\Theta(n^k)$ and B has size $\Theta(n^h)$ with $k \leq h$;*
   - *B is exponential-sized.*
2. *If A is polynomial-sized or finite then $A \leq_{au} B$ or $B \leq_{au} A$. If A is of size $\Theta(n^k)$, B is of size $\Theta(n^h)$ and $k < h$ then $A \leq_{au} B$ and $B \not\leq_{au} A$.*

Exercise 12.4. *Make an automatic one-one function which maps the domain $A = 0^*(1^* \cup 2^*)$ to a subset of $B = (0000)^*(1111)^*(2222)^*$, that is, show that $A \leq_{au} B$.*

The question on whether exponential-sized regular sets are always comparable with respect to *au*-reducibility was left open and is still unresolved.

Open Problem 12.5: Tan [85]. *Are there regular sets A, B such that $A \not\leq_{au} B$ and $B \not\leq_{au} A$?*

This open problem can be solved in the case that one considers context-free languages in place of regular languages. Then $A = \{x \cdot 2 \cdot y : x, y \in \{0, 1\}^*$ and $|x| = |y|\}$ and $B = \{0\}^*$. There is no automatic function mapping A to B in a one-one way, as A is exponential-sized and B is linear-sized. There is no automatic function mapping B to

A in a one-one way, as the range of this function would be an infinite regular set and all words in the language would have exactly one 2 in the middle which contradicts the regular pumping lemma. Hence these sets A and B are incomparable with respect to *au*-reducibility.

One might also look at reducibilities which are not automatic but still sufficiently easy. One of them is the self-concatenation mapping x to xx. There are two open questions related to this reduction.

Open Problem 12.6: Zhang [92]**.**

1. *Given a regular language A, is there a regular language B such that, for all x, $A(x) = B(xx)$?*

2. *Given a context-free language A, is there a context-free language B such that, for all x, $A(x) = B(xx)$?*

The converse direction is well-known, see, for example, Zhang [92]: If B is regular then the set $A = \{x : xx \in B\}$ is also regular. However, the set $B = \{0^n 1^n 2^m 0^m 1^k 2^k : n, m, k \in \mathbb{N}\}$ is context-free while the corresponding A given as

$$A = \{x : xx \in B\} = \{0^n 1^n 2^n : n \in \mathbb{N}\}$$

is not context-free; A is a standard example of a properly context-sensitive set.

Follow-up work by Fung [31] deals with the *xm*-reducibility. Here one maps x to $x \cdot x^{mi}$ where the function $x \mapsto x^{mi}$ maps an x to its mirror-image, so $(01122123)^{mi} = 32122110$. Now one can show that for every regular set A there is a regular set B such that $A(x) = B(x \cdot x^{mi})$. The set B is chosen as $\{u :$ there are an odd number of pairs (y, z) with $u = yz$ and $y \in A$ and $z \in A^{mi}\}$.

An *ordered group* $(G, +, <, 0)$ satisfies besides the group axioms also the order axioms, namely that $x < y \wedge y < z$ implies $x < z$ and that always exactly one of the three options $x < y$, $y < x$ and $x = y$. Furthermore, the group operation $+$ has to be compatible with the ordering $<$, that is, if $x < y$ then $x + z < y + z$ and $z + x < z + y$ for all x, y, z. Jain, Khoussainov, Stephan, Teng and Zou [46] showed that a fully automatic ordered group is always commutative. Furthermore, they investigated the following problem which was first posed by Khoussainov.

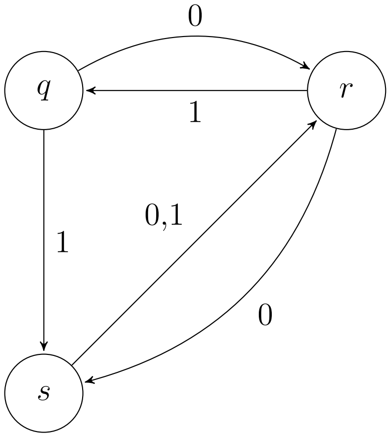
Open Problem 12.7: Khoussainov [46]**.** *Is there a fully automatic group $(G, +)$ isomorphic to the integers such that $A = \{x \in G : x$ is mapped to a positive number by the isomorphism$\}$ is not regular?*

Jain, Khoussainov, Stephan, Teng and Zou [46] showed that the corresponding question can be answered positively if one takes G to be isomorphic to the rationals with denominators being powers of 6: $G = \{n/6^m : n \in \mathbb{Z} \wedge m \in \mathbb{N}\}$. In this case one can represent the fractional parts as a sum of a binary represented part $n'/2^{m'}$ and ternary represented part $n''/3^{m''}$ and one can do addition on such a representation but one cannot compare the numbers with a finite automaton.

Second: Famous Open Problems. For a given dfa, a synchronising word w such that for all states q, the resulting state $\delta(q, w)$ is the same. Not every dfa has a synchronising word, for example the dfa which computes the remainder by 3 of a sequence of digits cannot have such a state. Černý investigated under which conditions a dfa has a synchronising word and if so, what the length of the shortest synchronising word is. He got the following main result.

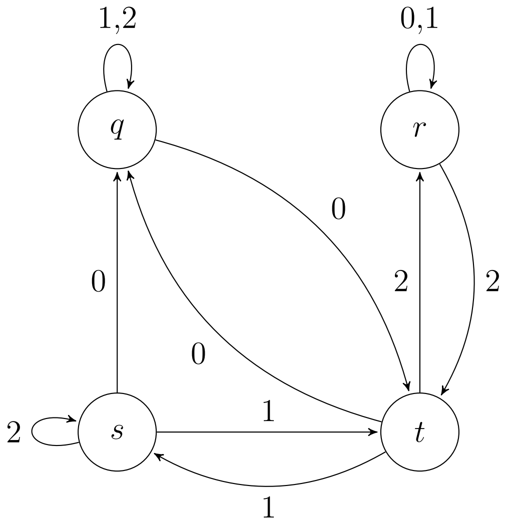
Theorem 12.8: Černý [16]. *For each n there is a complete dfa with n states which has a synchronising word of length $(n-1)^2$ and no shorter ones.*

Example 12.9. The following automaton gives a dfa for which synchronising words exist and have at least the length 4; note that it is not needed to designate any states as starting or accepting, as this does not matter for the question investigated.



Now the word 0110 is a synchronising word which sends all states to r. For ease of notation, let $\delta(Q, w) = \{\delta(p, w) : p \in Q\}$ for any set Q of states. Note that $\delta(\{q, r, s\}, 1) = \{q, r, s\}$, hence the shortest synchronising word has to start with 0. Now $\delta(\{q, r, s\}, 0) = \{r, s\}$. Note that $\delta(\{r, s\}, 0) = \{r, s\}$, hence the next symbol has to be a 1 in order to achieve something and the synchronising word starts with 01 and $\delta(\{q, r, s\}, 01) = \{q, r\}$. As $\delta(\{q, r\}, 1) = \{q, s\}$ and $\delta(\{q, r\}, 0) = \{r, s\}$, there is no synchronising word of length 3. However, $\delta(\{q, r, s\}, 0110) = \delta(\{q, s\}, 0) = \{r\}$ and 0110 is a shortest synchronising word.

The next example is a complete dfa with $n = 4$ and alphabet $\{0, 1, 2\}$ for which a synchronising word exist and each such word has at least length 9.



The synchronising word for this automaton is 012020120. Again it starts with a 0 and the next symbol has to be a 1 as all others leave the set of reached states the same. The next symbol must be a 2, as a 1 or 0 would undo the modification brought by 01, that is, $\delta(\{q,r,s,t\},010) = \delta(\{q,r,s,t\},011) = \delta(\{q,r,s,t\},0)$. After 012 one can again apply 0 in order to reduce the number of alive states to two: $\delta(\{q,r,s,t\},0120) = \{q,t\}$. Now the next two symbols are 20 in order to move one alive state away from q and one gets $\delta(\{q,t\},20) = \{r,t\}$. Now $\delta(\{r,t\},12) = \{s,t\}$ which is the only set of two alive states which can be mapped into one alive state. This is done by applying 0, so that in summary $\delta(\{q,r,s,t\},012020120) = \{q\}$.

Upper bounds on the length of the shortest synchronising word are also known, however most likely they are not optimal and there is still a considerable gap between the quadratic lower and cubic upper bound.

Theorem 12.10: Frankl [30]; Klyachko, Rystsov and Spivak [55]; Pin [71]. *Assume a complete dfa has n states and has a synchronising word. Then it has a synchronising word not longer than $(n^3 - n)/6$.*

In the following, a weaker form of this theorem is proven with an easier to prove cubic upper bound; this bound is weaker by a factor 3 plus a term of order $O(n)$. Let Q be the set of states. If one has two states q, r and a word w longer than $n(n+1)/2+1$ such that $\delta(\{q,r\},w)$ consists of a single state, then there must be a splitting of w into xyz with $y \neq \varepsilon$ such that either $\delta(\{q,r\},x) = \delta(\{q,r\},xy)$ or $\delta(\{q,r\},x)$ consists of a single state, as there are only $n(n+1)/2$ many different pairs of states. In both cases, $\delta(\{q,r\},xz)$ would also consist of a single state, so that w can be replaced by a shorter word. Therefore one can find, inductively, words

$w_1, w_2, \ldots, w_{n-1}$ such that $\delta(Q, w_1 w_2 \ldots w_m)$ has at most $n-m$ states and each w_m has at most length $n(n+1)/2+1$. Then the overall length of the synchronising word is at most $n(n^2-1)/2+n-1 = (n^3+n-2)/2$. For some small n it is known that Černý's Conjecture is true.

Example 12.11. If $n=3$ and the automaton has a synchronising word, then there is a synchronising word of length up to 4.

Proof. Let q, r, s be the states of the complete dfa. One can choose the first symbol of a synchronising word such that at least two states get synchronised. That is, $\delta(\{q,r,s\}, v) \subseteq \{q,r\}$ for a single-letter word v, where q, r, s are some suitable naming of the three states of the dfa. Now there are only three sets of two states, hence each set of two states reachable from $\{q,r\}$ can be reached in up to two symbols. Therefore, a shortest synchronising word w for $\{q,r\}$ must have the property that no set of states is repeated and therefore w has at most the length 3, that is, after the third symbol the corresponding set of alive states has only one element. Thus $\delta(\{q,r,s\}, vw)$ has one element and $|vw| \leq 4$. ▮

One can also show the conjecture for other small values of n; however, the full conjecture is still open.

Open Problem 12.12: Černý's Conjecture [16]. *Černý conjectured that if a complete dfa with n states has synchronising words, then the shortest such word has at most length $(n-1)^2$.*

Exercise 12.13. *Prove Černý's conjecture for $n=4$; that is, prove that given a complete dfa with four states which has a synchronising word, the shortest synchronising word for this dfa has at most the length 9.*

Another basic question in automata theory is that of the star height. If one permits only the basic operations of forming regular expressions, namely union, concatenation and Kleene star, one can introduce levels of star usage. Namely one does the following:

- Let S_0 contain all finite languages, note that S_0 is closed under union and concatenation;
- For each n, let S_{n+1} contain all languages which can be formed by taking unions and concatenations of languages of the form L or L^* with $L \in S_n$.

The star-height of a regular language L is the minimal n such that $L \in S_n$. Here are some examples.

- The language $L_0 = \{0, 11, 222, 3333\}$ is finite and has star-height 0;

- The language $L_1 = \{00, 11\}^*$ has star-height 1;
- The language $L_2 = (\{00, 11\}^* \cdot \{22, 33\} \cdot \{00, 11\}^* \cdot \{22, 33\})^*$ has star-height 2.

Eggan [24] investigated the star-height and provided a method to compute it from the possible nfas which recognise a regular language. It is known that there are infinitely many different levels of star-height a regular language can take. There is a generalisation, called the generalised star-height. A language is called star-free if it can be build from finite languages and Σ^* using union, intersection, set difference and concatenation. These languages are also called those of generalised star-height 0. The languages of generalised star-height $n+1$ are formed by all expressions obtained by starting with languages of star-height n and their Kleene star languages and then again combining them using union, intersection, set-difference and concatenation. Here examples for the first two levels:

- The language $\{0, 1\}^*$ has generalised star-height 0, as
$$\{0, 1\}^* = \Sigma^* - \bigcup_{a \in \Sigma - \{0,1\}} \Sigma^* a \Sigma^*;$$
- L_2 from above has generalised star-height 1, as
$$L_2 = \{00, 11, 22, 33\}^* \cap \{0, 1\}^* \cdot (\{22, 33\} \cdot \{0, 1\}^* \cdot \{22, 33\} \cdot \{0, 1\}^*)^*$$
and so L_2 is the intersection of two languages of generalised star-height 1;
- $L_3 = \{w : w$ does not have a substring of the form $v\}$ for a fixed v is of generalised star-height 0 as $L_3 = \Sigma^* - \Sigma^* \cdot v \cdot \Sigma^*$;
- $L_4 = \{w : w$ has an even number of $0\}$ is of generalised star-height 1.

It is unknown whether every regular language falls into one of these two levels.

Open Problem 12.14. *Are there any regular languages of generalised star-height* 2*? Is there a maximal* n *such that regular languages of generalised star-height* n *exist? If so, what is this* n*?*

Exercise 12.15. *Determine the generalised star-height of the following languages over the alphabet* $\{0, 1, 2\}$ *– it is zero or one:*

1. $\{00, 11, 22\}^* \cdot \{000, 111, 222\}^*$;
2. $\{0, 1\}^* \cdot 2 \cdot \{0, 1\}^* \cdot 2 \cdot \{0, 1\}^*$;
3. $(\{0, 1\}^* \cdot 2 \cdot \{0, 1\}^* \cdot 2 \cdot \{0, 1\}^*)^*$;
4. $(\{0, 1\}^* \cdot 2 \cdot \{0, 1\}^*)^*$;
5. $(\{0, 1\}^+ \cdot 22)^*$;

6. $(\{0,1\}^* \cdot 22)^*$;
7. $(((00)^+ \cdot 11)^+ \cdot 22)^+$.

In automatic groups one selects a subset G of words over the generators to represent all group elements. However, one mostly ignores the words not in G. A central question is how difficult the word problem is, that is, how difficult is it to determine whether a word over the generators (including the inverses) represents a word $w \in G$. That is, if Σ denotes the generators then the word problem is the set $\{(v,w) : v \in \Sigma^*, w \in G, v = w$ as group elements$\}$. One can show that the word problem can be solved in polynomial time (PTIME) by the algorithm which starts with the memory u being initialised as the neutral word ε and then reads out one symbol a after another from v and updates u to the member of G representing $u \cdot a$; these updates are all automatic and one has to just invoke the corresponding automatic function $|v|$ times. There is a complexity class LOGSPACE in which one permits the algorithm to use a work space of size logarithmic in the length of the input and to access the input with pointers pointing on some positions and permitting to read the symbol where they point to. These pointers can move forward or backward in the input, but not be moved beyond the beginning and end of the input. Now the algorithm can run arbitrary long but has to keep the memory constraint and at the end comes up with the answer ACCEPT or REJECT. Although there is no time constraint, one can show that the algorithm either needs polynomial time or runs forever, hence LOGSPACE is a subclass of PTIME. An open problem is whether the word problem of an automatic group can be solved in this subclass.

Open Problem 12.16. *Is the word problem of each automatic group solvable in LOGSPACE?*

Note that a negative answer to this problem would prove that LOGSPACE $\neq$ PTIME what might even be a more difficult open problem. On the other hand, a positive answer, that is, a LOGSPACE algorithm might be difficult to find, as people looked for it in vane for more than 30 years. So this could be a quite hard open problem.

Widely investigated questions in automata theory is the complexity of membership for the various levels of the Chomsky hierarchy. While for the level of regular language, the usage of dfa provides the optimal answer, the best algorithms are not yet known for the context-free and context-sensitive languages.

Open Problem 12.17. *What is the best time complexity to decide the membership of a context-free language?*

Open Problem 12.18. *Can the membership in a given context-sensitive language be decided in deterministic linear space?*

Both questions are algorithmically important. Cocke, Younger and Kasami provided an algorithm which run in $O(n^3)$ to decide the membership of context-free languages. Better algorithms were obtained using fast matrix multiplication and today bounds around $O(n^{2.38})$ are known. Concerning the context-sensitive membership problem, it is known to be possible in $O(n^2)$ space and nondeterministically in linear space; so the main question is whether this trade-off cen be reduced. These two problems are also quite hard, as any progress which involves the handling of fundamental complexity classes.

One topic much investigated in theoretical computer science is whether the isomorphism problem of certain structures are decidable and this had also been asked for automatic structures. For many possible structures, negative answers were found as the structures were too general. For example, Kuske, Liu and Lohrey [57] showed that it is undecidable whether two automatic equivalence relations are isomorphic. On the other hand, it is decidable whether a linear ordered set is isomorphic to the rationals: By the Theorem of Khoussainov and Nerode, one can decide whether sentences formulated using the ordering in first order logic are true and therefore one checks whether the following conditions are true: (a) There is no least element; (b) There is no greatest element; (c) Between any two elements there is some other element. If these are true, the corresponding linear order is dense and without end-points and therefore isomorphic to the ordering of the rationals, as automatic linear orders have always an at most countable domain. There are still some isomorphism problems for which it is not known whether they can be decided.

Open Problem 12.19. *Are there algorithms which decide the following questions, provided that the assumptions are met?*

1. *Assume that $(A, Succ_A, P_A)$ and $(B, Succ_B, P_B)$ are automatic structures such that $(A, Succ_A)$ and $(B, Succ_B)$ are isomorphic to the natural numbers with successor and that P_A and P_B are regular predicates (subsets) on A and B. Is $(A, Succ_A, P_A)$ isomorphic to $(B, Succ_B, P_B)$?*

2. *Assume that $(A, +)$ and $(B, +)$ are commutative fully automatic groups. Is $(A, +)$ isomorphic to $(B, +)$?*

An important open problem for parity games is the time complexity for finding the winner of a parity game, when both players play optimally; initially the algorithms took exponential time [63, 93]. Subsequently Petersson and Vorobyov [70] devised a subexponential randomised algorithm and Jurdziński, Paterson and Zwick [51] a deterministic algorithm of similar complexity; here the subexponential complexity was approximately $n^{O(\sqrt{n})}$. Furthermore, McNaughton [63] showed that the winner of a parity game can be determined in time $O(n^m)$, where n is the number of nodes and m the maximum value aka colour aka priority of the nodes. The following result provides an improved subexponential bound which is also in quasipolynomial time. For the below, it is assumed that in every node, a move can be made, so that the parity game never gets stuck. Furthermore, $\log(h) = \min\{k \in \{1, 2, 3, \ldots\} : 2^k \geq h\}$, so that the logarithm is always a non-zero natural number, what permits to use the logarithm in multiplicative expressions without getting 0 as well as indices in arrays.

Theorem 12.20: Calude, Jain, Khoussainov, Li, Stephan [11]. *One can decide in alternating polylogarithmic space which player has a winning strategy in a given parity game. When the game has n nodes and the values of the nodes are a subset of $\{1, 2, \ldots, m\}$ then the algorithm can do this in $O(\log(n) \cdot \log(m))$ alternating space.*

Proof. The idea of the proof is that the players move around a marker in the game as before; however, together with the move they update two winning statistics, one for Anke and one for Boris, such that whenever one player follows a memoryless winning strategy for the parity game then this winning statistic will mature (indicating a win for the player) while the winning statistic of the opponent will not mature (and thus not indicate a win for the opponent). It is known that every parity game has for one player a memoryless winning strategy, that is, the strategy tells the player for each node where to move next, independent of the history. The winning statistic of Anke has the following goal: to track whether the game goes through a cycle whose largest node is a node of Anke. Note that if Anke follows a memoryless winning strategy then the game will eventually go through a cycle and the largest node of any cycle the game goes through is always a node of Anke's parity; it will never be a node of Boris' parity, as then Anke's strategy would not be a memoryless winning strategy and Boris could repeat that cycle as often as he wants and thus obtain that a node of his parity is the limit superior of the play.

The naive method to do the tracking would be to archive the last $2n + 1$ nodes visited, however, this takes $O(n \cdot \log(n))$ space and would be too much for the intended result. Thus one constructs a winning statistic which still leads to an Anke win in the case that Anke plays a memoryless winning strategy, however, it will take longer time until it verifies that there was a loop with an Anke-node as largest member, as the winning statistic only memorises partial information due to space restrictions.

Nodes with even value are called Anke-nodes and nodes with an odd value are called Boris-nodes. For convenience, the following convention is made: when comparing nodes with "$<$" and "$\leq$", the corresponding comparison relates to the values of the nodes; when comparing them with "$=$" or "$\neq$", the corresponding comparison refers to the nodes themselves and different nodes with the same value are different with respect for this comparison. Furthermore, the value 0 is reserved for entries in winning statistics which are void and $0 < b$ for all nodes b.

In Anke's winning statistics, an i-sequence is a sequence of nodes $a_1, a_2, \ldots, a_{2^i}$ which had been observed within the course of the game such that, for each $k \in \{1, 2, \ldots, 2^i - 1\}$, the value $\max_{\leq}\{b : b = a_k \vee b = a_{k+1} \vee b$ was observed between a_k and $a_{k+1}\}$ has Anke's parity. For each i-sequence, the winning statistic does not store the sequence itself but it only stores the maximum value b_i of a node which either occurs as the last member of the sequence or occurs after the sequence.

The following invariants are kept throughout the game and are formulated for Anke's winning statistic, those for Boris' winning statistic are defined with the names of Anke and Boris interchanged:

- Only b_i with $0 \leq i \leq \log(n) + 3$ are considered and each such b_i is either zero or a value of an Anke-node or a value of a Boris-node;
- An entry b_i refers to an i-sequence which occurred in the play so far iff $b_i > 0$;
- If b_i, b_j are both non-zero and $i < j$ then $b_i \leq b_j$;
- If b_i, b_j are both non-zero and $i < j$ then they refer to an i-sequence and an j-sequence, respectively, and, in the play of the game, the i-sequence starts only after the value b_j was observed at or after the end of the j-sequence.

Both players' winning statistics are initialised with $b_i = 0$ for all i when the game starts. In each cycle, when the player whose turn is to move has chosen to move into the node with value b, the winning statistics of Anke and then of Boris are updated as follows, here the algorithm for Anke is given and it is followed by an algorithm for Boris with the names of the players interchanged everywhere.

- If b is either an Anke-node or $b > b_0$ then one selects the largest i such that
  (a) either b_i is not an Anke-node but all b_j with $j < i$ are Anke nodes and $(i > 0 \Rightarrow \max\{b_0, b\}$ is an Anke-node)
  (b) or $0 < b_i < b$

  and one updates $b_i = b$ and $b_j = 0$ for all $j < i$;
- If this update produces a non-zero b_i for any i with $2^i > 2n$ then the game terminates with Anke being declared winner.

The winning statistic of Boris is maintained and updated by the same algorithm, with the roles of Anke and Boris being interchanged in the algorithm. When both winning statistics are updated without a termination then the game goes into the next round by letting the corresponding player choose a move.

When updating Anke's winning statistic and the update can be done by case (a) then one can form a new i-sequence by putting the j-sequences for $j = i - 1, i -$

$2, \ldots, 1, 0$ together and appending the one-node sequence b which then has the length $2^i = 2^{i-1} + 2^{i-2} + \ldots + 2^1 + 2^0 + 1$; in the case that $i = 0$ this condition just says that one forms a 0-sequence of length 2^0 just consisting of the node b. Note that in the case $i > 0$ the value $\max\{b_0, b\}$ is an Anke-node and therefore the highest node between the last member a of the F_0-sequence and b has the value $\max\{b_h, b\}$ and is an Anke-node. Furthermore, for every $j < i-1$, for the last node a of the $j+1$-sequence and the first node a' of the j-sequence in the new i-sequence, the highest value of a node in the play between these two nodes a, a' is b_{j+1} which, by choice, has Anke's parity. Thus the overall combined sequence is an i-sequence replacing the previous sequences and b is the last node of this sequence and thus, currently, also the largest node after the end of the sequence. All j-sequences with $j < i$ are merged into the new i-sequence and thus their entries are set back to $b_j = 0$.

When updating Anke's winning statistic and the update can be done by case (b) then one only replaces the largest value at or after the end of the i-sequence (which exists by $b_i > 0$) by the new value $b > b_i$ and one discards all j-sequences with $j < i$ what is indicated by setting $b_j = 0$ for all $j < i$.

The same rules apply to the updates of Boris' winning statistics with the roles of Anke and Boris interchanged everywhere.

Note when updating Anke's winning statistic with a move to an Anke-node b, then one can always make an update of type (a) with i being the least number where b_i is not an Anke-node (which exists as the game would have terminated before otherwise). Similarly for updating Boris winning statistics.

If a player wins then the play contains a loop with its maximum node being a node of the player: Without loss of generality assume this winning player to be Anke. The game is won by an i-sequence being observed in Anke's winning statistics with $2^i > 2n$; thus some node occurs at least three times in the i-sequence and there are $h, \ell \in \{1, 2, \ldots, 2^i\}$ with $h < \ell$ such that the same player moves at a_h and a_ℓ and furthermore $a_h = a_\ell$ with respect to the nodes $a_1, a_2, \ldots, a_{F_i}$ of the observed i-sequence. The maximum value b' between a_h and a_ℓ in the play is occurring between some a_k and a_{k+1} (inclusively) for a k with $h \leq k < \ell$. Now, by definition of an i-sequence, b' has Anke's parity. Thus a loop has been observed for which the maximum node is an Anke node.

A player playing a memoryless winning strategy for parity games does not lose: If a player plays a memoryless winning strategy then the opponent cannot go into a loop where the maximum node is of the opponent's parity, as otherwise the opponent could cycle in that loop forever and then win the parity game, contradicting to the player playing a memoryless winning strategy. Thus, when a player follows a memoryless winning strategy, the whole play does not contain any loop where the opponent has the maximum node and so the opponent is during the whole play never declared to be the winner by the winning statistics.

A player playing a memoryless winning strategy for parity games will eventually win: For brevity assume that the player is Anke, the case of Boris is symmetric. The values b_i analysed below refer to Anke's winning statistic.

Assume that an infinite play of the game has the limit superior c which, by assumption, is an Anke-node. For each time t let

$$card(c,t) = \sum_{k:\ b_k(t) \text{ is an Anke-node and } b_k(t) \geq c} 2^k$$

where the $b_k(t)$ refer to the value of b_k at the end of step t. Now it is shown that whenever at times t, t' with $t < t'$ a move to c was made with c being an Anke-node and no move strictly between t, t' was to any node $c' \geq c$ then $card(c,t) < card(c,t')$. To see this, let i be the largest index where there is a step t'' with $t < t'' \leq t'$ such that b_i becomes updated in step t''. Now one considers several cases:

- Case $b_i(t'') = 0$: This case does only occur if also b_{i+1} gets updated and contradicts the choice of i, so it does not need to be considered.
- Case $b_i(t) \geq c$ and $b_i(t)$ is an Anke node: In this case, the only way to update this node at t'' is to do an update of type (a) and then also the entry $b_{i+1}(t'')$ would be changed in contradiction of the choice of i, so this case also does not need to be considered.
- Case $b_i(t)$ is a Boris node and $b_i(t) \geq c$: Then an update is possible only by case (a). If $b_i(t'') < c$ then, at step t', another update will occur and enforce by (b) that $b_i(t') = c$. The value $card(c,t)$ is largest when all $b_j(t)$ with $j < i$ are Anke-nodes at step t and even in this worst case it holds that $card(c,t') - card(c,t) \geq 2^i - \sum_{j:j<i} 2^j \geq 1$.
- Case $0 < b_i(t) < c$: Then latest at stage t', as an update of type (b) at i is possible, it will be enforced that $b_i(t') = c$ while $b_j(t) < c$ for all $j \leq i$ and therefore $card(c,t') \geq card(c,t) + 2^i \geq card(c,t) + 1$.
- Case $b_i(t) = 0$: Then at stage t'' an update of type (a) will make $b_i(t'') > 0$ and, in the case that $b_i(t'') < c$, a further update of type (b) will at stage t' enforce that $b_i(t') = c$. Again, the value $card(c,t)$ is largest when all $b_j(t)$ with $j < i$ are Anke-nodes at step t and even in this worst case it holds that $card(c,t') - card(c,t) \geq 2^i - \sum_{j:j<i} 2^j \geq 1$.

Thus, once all moves involving nodes larger than c have been done in the play, there will still be infinitely many moves to nodes of value c and for each two subsequent such moves at t, t' it will hold that $card(c,t) + 1 \leq card(c,t')$. As a consequence, the number $card(c,t)$ for these nodes will, for sufficiently large t where a move to c is made, rely on some i with $b_i(t) \geq c$ and $2^i > 2n$ and latest then the termination condition of Anke will terminate the game with a win for Anke.

Thus, an alternating Turing machine can simulate both players and it will accept the computation whenever Anke has a winning strategy for the game taking the winning statistics into account. Thus the alternating Turing machine with space usage of $O(\log(n) \cdot \log(m))$ can decide whether the game, from some given starting point, will end up in Anke winning or in Boris winning, provided that the winner plays a memoryless winning strategy for the corresponding parity game (which always exists when the player can win the parity game). ∎

Chandra, Kozen and Stockmeyer [15] showed that everything what can be computed by an alternating Turing machine in polylogarithmic space can also be computed deterministically in quasipolynomial time. More precisely, their more precise bounds give that the running time of a deterministic Turing machine for the above mentioned problem is $O(n^{c\log(m)})$ for some constant c.

Theorem 12.21: Calude, Jain, Khoussainov, Li, Stephan [11]. *Assume that a parity game has n nodes which take values from $\{1, 2, \ldots, m\}$, note that one can always choose $m \leq n+1$. Now one can decide in time $O(n^{c\log(m)})$ which player has a winning strategy in the parity game.*

In some special cases with respect to the choice of m in dependence of n, one can obtain a polynomial time bound. McNaughton [63] showed that for every constant m, one can solve a parity game with n nodes having values from $\{1, 2, \ldots, m\}$ in time $O(n^m)$; Schewe [79, 81] and others brought down the bound, but it remained dependent on m. The next result shows that for fixed m and large n one can determine the winner of the parity game in $O(n^{5.04})$; the bound is, however, more general: If $m \leq h \cdot \log(n)$ then one can determine the winner of a parity game in $O(h^4 \cdot n^{3.45+\log(h+2)})$. This implies that one can solve the parity games in $O((16n)^{3.45+\log(\lceil m/\log(n)\rceil+2)})$. Calude, Jain, Khoussainov, Li and Stephan [11] give a slightly better bound for $h = 1$.

Theorem 12.22. *If $m \leq h \cdot \log(n)$ and $h \in \mathbb{N}$ then one can solve the parity game with n nodes which have values from $\{1, 2, \ldots, m\}$ in time $O(h^4 \cdot n^{3.45+\log(h+2)})$.*

Proof. Note that Theorem 12.20 actually showed that the following conditions are equivalent:

- Anke can win the parity game;

- Anke can play the parity game such that her winning statistic matures while Boris' winning statistic does not mature.

Thus one can simplify this and play a survival game with the following property: Anke wins the game iff the parity game runs forever without Boris achieving a win according to his winning statistics. If Boris follows a memoryless winning strategy for the parity game then Anke loses, if Anke follows a memoryless winning strategy for the parity game then she wins. Thus it is sufficient to track only Boris' winning statistics for the game. Thus Anke has a winning strategy for the parity game iff she has a winning strategy for the following survival game:

- The set Q of nodes of the survival game consists of nodes of the form $(a, p, \tilde{b})$ where a is a node of the parity game, the player $p \in \{\text{Anke}, \text{Boris}\}$ is that player whose turn is to move next and $\tilde{b}$ represents the winning statistic of Boris;
- Anke can move from $(a, \text{Anke}, \tilde{b})$ to $(a', \text{Boris}, \tilde{b}')$ iff she can move from a to a' in the parity game and this move causes the winning statistic of Boris to be updated from $\tilde{b}$ to $\tilde{b}'$;
- Boris can move from $(a, \text{Boris}, \tilde{b})$ to $(a', \text{Anke}, \tilde{b}')$ iff he can move from a to a' in the parity game and this move causes the winning statistic of Boris to be updated from $\tilde{b}$ to $\tilde{b}'$;
- The starting node is $(s, \text{Anke}, \tilde{0})$ where $\tilde{0}$ is the vector of all b_i being 0 and s is the starting node of the parity game.

To estimate the number of members of Q, first one codes Boris' winning condition $b_0, b_1, \ldots, b_{\lceil \log(n) \rceil + 2}$ by a new sequence $\hat{b}_0, \hat{b}_1, \ldots, \hat{b}_{\lceil \log(n) \rceil + 2}$ as follows: $\hat{b}_0 = b_0$ and, for all $i < \lceil \log(n) \rceil + 2$, if $b_{i+1} = 0$ then $\hat{b}_{i+1} = \hat{b}_i + 1$ else $\hat{b}_{i+1} = \hat{b}_i + 2 + \min\{b_{i+1} - b_j : j \leq i\}$. Note that the latter just says that $b_{i+2} = \hat{b}_i + 2 + (b_i - b_j)$ for the most recent j where $b_j \neq 0$. Now $\hat{b}_{\lceil \log(n) \rceil + 2} \leq 2 \cdot (\lceil \log(n) \rceil + 2) + h \cdot b_{\lceil \log(n) \rceil + 2} \leq (h+2) \cdot (\lceil \log(n) \rceil + 3)$ what gives $O(n^{h+2})$. Thus the number of possible values of the winning statistics can all be coded with $(h+2) \cdot (\lceil \log(n) \rceil + 3)$ bits. However, one can get a better value by observing that only $\lceil \log(n) \rceil + 3$ of these bits are 1. The number of all ways to choose $\lceil \log(n) \rceil + 3$ out of $(h+2) \cdot (\lceil \log(n) \rceil + 3)$ numbers can, by the Wikipedia page on binomial coefficients and the inequality using the entropy in there, be bounded by

$$\begin{array}{rl} & 2^{(\log(n)+4) \cdot (h+2) \cdot ((1/(h+2)) \cdot \log(h+2) + ((h+1)/(h+2)) \cdot \log((h+2)/(h+1)))} \\ = & 2^{(\log(n)+4) \cdot (\log(h+2) + \log(1+1/(h+1)) \cdot (h+1))} \\ = & (16n)^{\log(h+2) + (\log(1+1/(h+1)) \cdot (h+1))} \\ \leq & (16n)^{1.45 + \log(h+2)} \leq c \cdot h^4 \cdot n^{1.45 + \log(h+2)} \end{array}$$

for some constant c, so the whole expression is in $O(h^4 \cdot n^{1.45 + \log(h+2)})$. In these equations, it is used that $\log(1 + 1/(h+1)) \cdot (h+1) \leq \log(2.718282) \leq 1.45$, for all

$h \in \mathbb{N}$, where 2.71828 is an upper bound of Euler's number. Furthermore, one has to multiply this by one n and by 2 in order to store the current player and current position, so in total Q has size $O(h^4 \cdot n^{2.45+\log(h+2)})$ and the remaining part of the proof will show that the runtime is bounded by $O(h^4 \cdot n^{3.45+\log(h+2)})$.

The survival game can be decided in $O(|Q| \cdot n)$: The algorithm would be the following: First one computes for each node $q \in Q$ the list of the up to n successors and also generates a linked list of predecessors such that the collection of all these lists together has the length $|Q| \cdot n$. These inverted lists can also be generated in time $O(|Q| \cdot n)$. Furthermore, one can determine a list of $Q' \subseteq Q$ of nodes where Boris winning statistic has matured (that is, Boris has won); determining these nodes is also in time $O(|Q|)$.

Note that a node is a winning node for Boris if either Anke moves from this node and all successor nodes are winning nodes for Boris or Boris moves from this node and some successor is a winning node for Boris. This idea will lead to the algorithm below.

For this, a tracking number k_q is introduced which is maintained such that the winning nodes for Boris will eventually all have $k_q = 0$ and that k_q indicates how many further times one has to approach the node until it can be declared a winning node for Boris. The numbers k_q are initialised by the following rule:

- On nodes $q \in Q'$ the number k_q is 1;
- On nodes $q = (a, \text{Anke}, \tilde{b}) \notin Q'$, the number k_q is initialised as the number of nodes q' such that Anke can move from q to q';
- On nodes $q = (a, \text{Boris}, \tilde{b}) \notin Q'$, the number k_q is initialised as 1;

These numbers can be computed from the length of the list of predecessors of q for each $q \in Q$. Now one calls the following recursive procedure initially for all $q \in Q'$ and each call updates the number k_q. The recursive call does the following:

- If $k_q = 0$ then return without any further action else update $k_q = k_q - 1$;
- If after this update still $k_q > 0$ then return without further action;
- Otherwise, that is when k_q originally was 1 when entering the call then call recursively all predecessors q' of q with the same algorithm.

After the termination of all these recursive calls, one looks at k_q for the start node q of the survival game. If $k_q > 0$ then Anke wins else Boris wins.

Note that in this algorithm, for each node $q \in Q$ the predecessors are only called at most once, namely when k_q goes down from 1 to 0 and that is the time where it is determined that the node is a winning node for Boris. Thus there are at most

$O(|Q| \cdot n)$ many recursive calls and the overall complexity is $O(|Q| \cdot n)$.

For the verification, the main invariant is that k_q originally says for how many of the successors of q one must check that they are winning nodes for Boris until one can conclude that the node q is also a winning node of Boris. In the case that the winning statistics of Boris have matured in the node q, the value k_q is taken to be 1 so that the node is processed once with all the recursive calls in the recursive algorithm. For nodes where it is Boris' turn to move, there needs also be only one outgoing move which produces a win of Boris. Thus one initialises k_q as 1 and as soon as this outgoing node is found, k_q goes to 0 what means that the node is declared a winning node for Boris. In the case that the node q is a node where Anke moves then one has to enforce that Anke has no choice but to go to a winning node for Boris. Thus k_q is initialised as the number of moves which Anke can move in this node and each time when one of these successor nodes is declared a winning node for Boris, k_q goes down by one. Note that once the recursive algorithm is completed for all nodes, exactly the nodes with $k_q = 0$ are the winning nodes of Boris in this survival game. ▮

For the special case of $h = 1$, the more direct bound $O(n^{h+4})$ is slightly better than the derived bound of $O(n^{3.45+\log(3)})$; however, in the general case of larger h, the bound $O(n^{3.45+\log(h+2)})$ is better.

When considering $h = 1$, this special case shows that, for each constant m, the parity game with n nodes having values from $\{1, 2, \ldots, m\}$ can be solved in time $O(n^5) + g(m)$ for some function g. Such problems are called "Fixed Parameter Tractable", as for each fixed parameter m the corresponding algorithm runs in polynomial time and this polynomial is the same for all m, except for the additive constant $g(m)$ depending on m. Downey and Fellows [22] provide an introduction to the field of parameterised complexity.

Exercise 12.23. *Show that one can decide, for all sufficiently large m, n, the parity games with n nodes and values from $\{1, 2, \ldots, m\}$ in time $O(n^{\log(m)+20})$; for this use a direct coding of the winning conditions with $\lceil \log(n) + 3 \rceil \cdot \lceil \log(m) + 1 \rceil$ bits rather than the above methods with the binomial coefficients. Furthermore, show that the memoryless winning-strategy of the winner can then be computed with the same time bound (the constant 20 is generous enough).*

Open Problem 12.24. Is there a polynomial time algorithm (in the number n of nodes of the parity game) to decide which player would win the parity game?

Exercise 12.25. *Let $A = \{0\}^* \cdot \{1\}^*$, $B = \{00\}^* \cdot \{11\}^* \cdot \{22\}^*$; $C = \{00, 11, 22\}^*$; $D = \{0, 1\}^* - \{1\}^*$. How are the above sets A, B, C, D ordered by $\leq_{au}$? Provide the reductions where they exist.*

Exercise 12.26. *For the sets A, B, C, D from Exercise 12.25, how are they ordered by $\leq_{tr}$?*

Exercise 12.27. *For the sets A, B, C from Exercise 12.25, determine regular sets A', B', C' such that for all x, $x \in A \Leftrightarrow xx \in A'$ and $x \in B \Leftrightarrow xx \in B'$ and $x \in C \Leftrightarrow xx \in C'$.*

Exercise 12.28. *Provide a regular set E such that there is a regular E' satisfying $\forall x\,[x \in E \Leftrightarrow xx \in E']$ but this E' is neither E nor $E \cdot E$.*

Exercise 12.29. *Let $F = (\{0\}^* \cdot \{1\} \cdot \{0\}^* \cdot \{1\})^*$. Determine for F the minimal complete dfa and either determine its smallest synchronising word or show that it does not exist.*

Exercise 12.30. *Let $G = (\{0\}^+ \cdot \{1\} \cdot \{0\}^+ \cdot \{1\})^*$. Determine for G the minimal complete dfa and either determine its smallest synchronising word or show that it does not exist.*

Exercise 12.31. *Let $H = (\{0\}^+ \cdot \{1\} \cdot \{0\}^+ \cdot \{1\})^* \cup (\{0,1\}^* \cdot \{11\} \cdot \{0,1\}^* \cdot \{00\}) \cup (\{1\} \cdot \{0,1\}^* \cdot \{00\})$. Determine for H the minimal complete dfa and either determine its smallest synchronising word or show that it does not exist.*

Selftest 12.32. *Construct a learning algorithm for the class of all regular languages which uses only equivalence queries such that the number of queries is linear in the sum consisting of the number of states of a dfa for the target and the number of symbols in the longest counter example seen.*

Selftest 12.33. *Let* $I = \{0,1\}^*$ *and for all* $e \in I$, $L_e = \{x \in \{0,1\}^* : x <_{lex} e\}$. *Is this automatic family learnable from positive data?*

If the answer above is "yes" then describe how the learner works; if the answer above is "no" then explain why a learner does not exist.

Selftest 12.34. *Assume that an automatic representation of the ordinals strictly below* ω^n *is given for some positive natural number* n*; here not only the order but also the ordinal addition is fully automatic. Now consider the class of all sets* $L_{\alpha,\beta} = \{\gamma < \omega^n : \alpha \leq \gamma < \beta\}$, *where* α *and* β *are ordinals chosen such that the set* $L_{\alpha,\beta}$ *is closed under ordinal addition, that is, when* $\gamma, \gamma' \in L_{\alpha,\beta}$ *so is* $\gamma + \gamma'$. *If this class is learnable then provide an automatic an automatic learner (using some automatic family representing the class as hypothesis space) else explain why the class is not learnable.*

Selftest 12.35. *Assume that a dfa has states and alphabet* $\{0,1,\ldots,9\}$ *and the successor of state* a *on symbol* b *is defined as follows: If* $a < b$ *then the successor is* $b - a - 1$ *else the successor is* $a - b$.

Determine whether this dfa has a synchronising word, that is, a word which maps all states to the same state. If so then write-down a synchronising word which is as short as possible else explain why there is no synchronising word.

Solution for Selftest 12.32. The idea is to "cheat" by forcing the teacher to output long counterexamples. So one takes a list of all deterministic finite automata dfa_0, dfa_1, ... and computes for each dfa_n and automaton dfa'_n such that

- dfa'_n has at least n states and there is no dfa with less states for the same language;
- the language recognised by dfa'_n differs from the language recognised by dfa_n by exactly one word of at least length n.

This can be achieved by searching for the first $m \geq n$ such that the minimal automaton for the language obtained by taking the symmetric difference of $\{0^m\}$ and the language recognised by dfa_n needs at least n states. The automata dfa'_n can be computed from dfa_n in polynomial time. Now the algorithm does the following:

1. Let $n = 0$;
2. Compute dfa_n and dfa'_n;
3. Ask if dfa'_n is correct;
4. If answer is "yes" then conjecture dfa'_n and terminate;
5. Ask if dfa_n is correct;
6. If answer is "yes" then conjecture dfa_n and terminate;
7. Let $n = n + 1$ and go to step 2.

Note that in this algorithm, if the language to be learnt turns out to be the one generated by dfa'_n then the number of states of the dfa is at least n and only $2n - 1$ queries had been made until learning success; if the language to be learnt turns out to be the one generated by dfa_n then dfa'_n had been asked before, differing from the language of dfa_n by exactly one word which has length n or more and at only $2n$ queries had been made, again the complexity bound is kept.

Solution for Selftest 12.33. The answer is "no". Assume by way of contradiction, that there is a learner M. Then $L_1 = \{\varepsilon\} \cup 0 \cdot \{0, 1\}^*$. By Angluin's tell-tale condition, there is a finite subset F of L_1 such that there should be no set L_u with $F \subseteq L_u \subset L_1$. Given such an F, let n be the length of the longest word in F and consider L_{01^n}. All members of L_1 up to length n satisfy that they are lexicographically strictly before 01^n and thus $F \subseteq L_{01^n} \subset L_1$. Thus, F cannot be a tell-tale set for L_1 and the class cannot be learnable by Angluin's tell-tale criterion.

Solution for Selftest 12.34. If $\gamma \in L_{\alpha,\beta}$ then also $\gamma + \gamma$, $\gamma + \gamma + \gamma$ and so on are in $L_{\alpha,\beta}$, furthermore, if $\omega^k \leq \gamma < \omega^{k+1}$ then all numbers between γ and ω^{k+1} (excluding ω^{k+1} itself) must be in $L_{\alpha,\beta}$; in the case that $\gamma = 0$, $\omega^{m(\gamma)} = 1$. Thus β is of the form

ω^m for some m with $0 \leq m \leq n$. So the learning algorithm does the following:

For each datum γ the learner computes $m(\gamma)$ to be the first ω-power strictly above γ, that is, $\gamma < \omega^{m(\gamma)} \leq \omega^m$ for all m with $\gamma < \omega^m$. The learner conjectures $\emptyset$ until some datum $\gamma \neq \#$ is observed. Then the learner let $\alpha = \gamma$ and $\beta = \omega^{m(\gamma)}$. At every further datum γ, α is replaced by $\min\{\alpha, \gamma\}$ and β is replaced by $\max\{\beta, \omega^{m(\gamma)}\}$.

This learner is automatic, as one can choose an automatic structure representing all ordinals up to ω^n together with their order; there are only finitely many ordinals of the form ω^m with $0 \leq m \leq n$, these can be archived and one can just define $\omega^{m(\gamma)}$ to be the least strict upper bound of γ from this finite list. Furthermore, the learner converges to a final hypothesis $L_{\alpha,\beta}$, as the minimum α of the language is seen after finite time and as the strict upper bound β, by being from a finite list, can only be updated a finite number of times to a larger number.

Solution for Selftest 12.35. The dfa has a synchronising word of length 4.

For getting a lower bound on the length, one can see that at most two states are mapped to the same state by a symbol b. So the first symbol maps the given 10 states to at least 5 different states, the next symbol maps these 5 states to at least 3 different states, the third symbol maps these 3 states to at least 2 states and the fourth symbol then might, perhaps, map the two states to one state. So the length of each synchronising word is at least 4.

Now consider the word 5321. The symbol 5 maps the states $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ to $\{0, 1, 2, 3, 4\}$, the symbol 3 maps $\{0, 1, 2, 3, 4\}$ to $\{0, 1, 2\}$, the symbol 2 maps $\{0, 1, 2\}$ to $\{0, 1\}$ and the symbol 1 maps $\{0, 1\}$ to $\{0\}$. Hence 5321 is a shortest synchronising word.

References

[1] Wilhelm Ackermann (1928). Zum Hilbertschen Aufbau der reellen Zahlen. *Mathematische Annalen*, 99:118–133, 1928.

[2] Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.

[3] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[4] Lenore Blum and Manuel Blum. Towards a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.

[5] Achim Blumensath and Erich Grädel. Automatic structures. *15th Annual IEEE Symposium on Logic in Computer Science*, LICS 2000, pages 51–62, 2000.

[6] Henrik Björklund, Sven Sandberg and Sergei Vorobyov. *On fixed-parameter complexity of infinite games.* Technical report 2003-038, Department of Information Technology, Uppsala University, Box 337, SE-751 05 Uppsala, Sweden.

[7] Henrik Björklund, Sven Sandberg and Sergei Vorobyov. Memoryless determinacy of parity and mean payoff games: a simple proof. *Theoretical Computer Science*, 310(1–3):365–378, 2004.

[8] Janusz A. Brzozowski. Derivatives of regular expressions. *Journal of the Association of Computing Machinery*, 11:481–494, 1964.

[9] J. Richard Büchi. On a decision method in restricted second order arithmetic. *Proceedings of the International Congress on Logic, Methodology and Philosophy of Science*, Stanford University Press, Stanford, California, 1960.

[10] J. Richard Büchi and Lawrence H. Landweber. Definability in the monadic second order theory of successor. *The Journal of Symbolic Logic*, 34:166–170, 1966.

[11] Cristian Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li and Frank Stephan. Deciding parity games in quasipolynomial time. *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, Montreal, QC, Canada, June 19-23, 2017. Pages 252–263, ACM, 2017.

[12] John Case, Sanjay Jain, Trong Dao Le, Yuh Shin Ong, Pavel Semukhin and Frank Stephan. Automatic learning of subclasses of pattern languages. *Information and Computation*, 218:17–35, 2012.

[13] John Case, Sanjay Jain, Samuel Seah and Frank Stephan. Automatic functions, linear time and learning. *Logical Methods in Computer Science*, 9(3), 2013.

[14] Christopher Chak, Rūsinņš Freivalds, Frank Stephan and Henrietta Tan. On block pumpable languages. *Theoretical Computer Science*, 609:272–285, 2016.

[15] Ashok K. Chandra, Dexter C. Kozen and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[16] Jan Černý. Poznámka k homogénnym experimentom s konečnými automatami. *Matematicko-fyzikálny Časopis Slovenskej Akadémie Vied*, 14:208–216, 1964. In Slovak. See also `http://en.wikipedia.org/wiki/Synchronizing_word`.

[17] Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124, 1956.

[18] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.

[19] John Cocke and Jacob T. Schwartz. *Programming languages and their compilers: Preliminary notes.* Technical Report, Courant Institute of Mathematical Sciences, New York University, 1970.

[20] Elias Dahlhaus and Manfred K. Warmuth. Membership for Growing Context-Sensitive Grammars Is Polynomial. *Journal of Computer and System Sciences*, 33:456–472, 1986.

[21] Christian Delhommé. Automaticité des ordinaux et des graphes homogènes. *Comptes Rendus Mathematique*, 339(1):5–10, 2004.

[22] Rodney G. Downey and Michael R. Fellows. *Parameterised Complexity.* Springer, Heidelberg, 1999.

[23] A. Ross Eckler. Leigh Mercer, Palindromist. *Word Ways*, 24(3):131–138, 1991.

[24] Lawrence C. Eggan. Transition graphs and the star-height of regular events. *Michigan Mathematical Journal*, 10(4):385–397, 1963.

[25] Andrzej Ehrenfeucht, Rohit Parikh and Grzegorz Rozenberg. Pumping lemmas for regular sets. SIAM Journal on Computing, 10:536–541, 1981.

[26] Andrzej Ehrenfeucht and Grzegorz Rozenberg. On the separating power of EOL systems. *RAIRO Informatique théorique* 17(1): 13–22, 1983.

[27] Andrzej Ehrenfeucht and H. Paul Zeiger. Complexity measures for regular expressions. *Journal of Computer and System Sciences*, 12(2):134–146, 1976.

[28] David Epstein, James Cannon, Derek Holt, Silvio Levy, Michael Paterson and William Thurston. *Word Processing in Groups.* Jones and Bartlett Publishers, Boston, Massachusetts, 1992.

[29] Robert W. Floyd and Donald E. Knuth. Addition machines. *SIAM Journal on Computing*, 19(2):329–340, 1990.

[30] Péter Frankl. An extremal problem for two families of sets. *European Journal of Combinatorics* 3:125–127, 1982.

[31] Dennis Fung. Automata Theory: The XM Problem. BComp Dissertation (Final Year Project), School of Computing, National University of Singapore, 2014.

[32] Jakub Gajarský, Michael Lampis, Kazuhisa Makino, Valia Mitsou and Sebastian Ordyniak. Parameterised algorithms for parity games. *Mathematical Foundations of Computer Science*, MFCS 2015. *Springer LNCS* 9235:336–347, 2015.

[33] William I. Gasarch. Guest Column: The second P =? NP Poll. *SIGACT News Complexity Theory Column*, 74, 2012.
`http://www.cs.umd.edu/~gasarch/papers/poll2012.pdf`

[34] Wouter Gelade and Frank Neven. Succinctness of the complement and intersection of regular expressions. *ACM Transactions in Computational Logic*, 13(1):4, 2012.

[35] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I, *Monatshefte für Mathematik und Physik*, 38: 173–198, 1931.

[36] Mark Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[37] Sheila Greibach. A new normal-form theorem for context-free phrase structure grammars. *Journal of the Association of Computing Machinery*, 12(1):42–52, 1965.

[38] Juris Hartmanis. Computational complexity of one-tape Turing machine computations. *Journal of the Association of Computing Machinery* 15:411–418, 1968.

[39] Juris Hartmanis and Janos Simon. On the power of multiplication in random access machines. *Fifteenth Annual Symposium on Switching and Automata Theory*, SWAT 1974, pages 13–23, IEEE, 1974.

[40] David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, to appear.
https://hal.archives-ouvertes.fr/hal-02070778v2/document.

[41] Bernard R. Hodgson. *Théories décidables par automate fini.* Ph.D. thesis, University of Montréal, 1976.

[42] Bernard R. Hodgson. Décidabilité par automate fini. *Annales des sciences mathématiques du Québec*, 7(1):39–57, 1983.

[43] John E. Hopcroft, Rajeev Motwani and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Third Edition, Addison-Wesley Publishing, Reading Massachusetts, 2007.

[44] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[45] Jeffrey Jaffe. A necessary and sufficient pumping lemma for regular languages. *ACM SIGACT News*, 10(2):48–49, 1978.

[46] Sanjay Jain, Bakhadyr Khoussainov, Frank Stephan, Dan Teng and Siyuan Zou. On semiautomatic structures. Computer Science – Theory and Applications – Ninth International *Computer Science Symposium in Russia*, CSR 2014, Moscow, Russia, June 7–11, 2014. Proceedings. Springer LNCS 8476:204–217, 2014.

[47] Sanjay Jain, Qinglong Luo and Frank Stephan. Learnability of automatic classes. *Language and Automata Theory and Applications*, Fourth International Conference, LATA 2010, Trier, May 2010, Proceedings. Springer LNCS 6031:293–307, 2010.

[48] Sanjay Jain, Yuh Shin Ong, Shi Pu and Frank Stephan. On automatic families. *Proceedings of the 11th Asian Logic Conference*, ALC 2009, in Honour of Professor Chong Chitat's 60th birthday, pages 94–113. World Scientific, 2011.

[49] Sanjay Jain, Yuh Shin Ong and Frank Stephan. Regular patterns, regular languages and context-free languages. *Information Processing Letters* 110:1114–1119, 2010.

[50] Marcin Jurdziński. Deciding the winner in parity games is in $\mathbf{UP} \cap \mathbf{Co} - \mathbf{UP}$. *Information Processing Letters*, 68(3):119–124, 1998.

[51] Marcin Jurdziński, Mike Paterson and Uri Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM Journal on Computing*, 38(4):1519–1532, 2008.

[52] Tadao Kasami. *An efficient recognition and syntax-analysis algorithm for context-free languages.* Technical Report, Air Force Cambridge Research Laboratories, 1965.

[53] Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. *Logical and Computational Complexity*, (International Workshop LCC 1994). Springer LNCS 960:367–392, 1995.

[54] Bakhadyr Khoussainov and Anil Nerode. *Automata Theory and its Applications.* Birkhäuser, 2001.

[55] A.A. Klyachko, Igor K. Rostsov and M.A. Spivak. An extremal combinatorial problem associated with the bound on the length of a synchronizing word in an automaton. *Cybernetics and Systems Analysis / Kibernetika*, 23(2):165–171, 1987.

[56] Lars Kristiansen and Juvenal Murwanashyaka. Decidable and undecidable fragments of first-order concatenation theory. *Fourteenth Conference on Computability in Europe*, CiE 2018, Kiel, Germany, *Springer LNCS* 10936:244-253, 2018. See also https://arxiv.org/abs/1804.06367.

[57] Dietrich Kuske, Jiamou Liu and Markus Lohrey. The isomorphism problem on classes of automatic structures with transitive relations. *Transactions of the American Mathematical Society*, 365:5103–5151, 2013.

[58] Roger Lyndon and Marcel-Paul Schützenberger. The equation $a^M = b^N c^P$ in a free group. *Michigan Mathematical Journal*, 9:289–298, 1962.

[59] Yuri V. Matiyasevich. Diofantovost' perechislimykh mnozhestv. *Doklady Akademii Nauk SSSR*, 191:297-282, 1970 (Russian). English translation: Enumerable sets are Diophantine, *Soviet Mathematics Doklady*, 11:354-358, 1970.

[60] Yuri Matiyasevich. *Hilbert's Tenth Problem.* MIT Press, Cambridge, Massachusetts, 1993.

[61] Kenneth Manders and Leonard Adleman. NP-complete decision problems for quadratic polynomials. *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, STOC 1976, pages 23–29, 1976.

[62] Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

[63] Robert McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

[64] George H. Mealy. A method to synthesizing sequential circuits. *Bell Systems Technical Journal*, 34(5):1045–1079, 1955.

[65] Albert R. Meyer and Michael J. Fischer. Economy of description by automata, grammars, and formal systems. *Twelfth Annual Symposium on Switching and Automata Theory*, SWAT 1971, pages 188–191, 1971.

[66] Edward F. Moore. Gedanken Experiments on sequential machines. *Automata Studies*, edited by C.E. Shannon and John McCarthy, Princeton University Press, Princeton, New Jersey, 1956.

[67] Anil Nerode. Linear automaton transformations. *Proceedings of the AMS*, 9:541–544, 1958.

[68] Maurice Nivat. Transductions des langages de Chomsky. *Annales de l'institut Fourier*, Grenoble, 18:339–455, 1968.

[69] William Ogden. A helpful result for proving inherent ambiguity. *Mathematical Systems Theory*, 2:191–194, 1968.

[70] Viktor Petersson and Sergei G. Vorobyov. A randomized subexponential algorithm for parity games. *Nordic Journal of Computing*, 8:324–345, 2001.

[71] Jean-Éric Pin. On two combinatorial problems arising from automata theory. *Annals of Discrete Mathematics*, 17:535–548, 1983.

[72] Michael O. Rabin and Dana Scott. Finite Automata and their Decision Problems, *IBM Journal of Research and Development*, 3:115–125, 1959.

[73] Frank P. Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, 30:264–286, 1930.

[74] Henry Gordon Rice. Classes of enumerable sets and their decision problems. *Transactions of the American Mathematical Society* 74:358–366, 1953.

[75] Rockford Ross and Karl Winklmann. Repetitive strings are not context-free. *RAIRO Informatique théorique* 16(3):191–199, 1982.

[76] Shmuel Safra. On the complexity of ω-automata. Proceedings twenty-ninth IEEE Symposium on Foundations of Computer Science, pages 319-327, 1988.

[77] Shmuel Safra. Exponential determinization for omega-Automata with a strong fairness acceptance condition. *SIAM Journal on Computing*, 36(3):803–814, 2006.

[78] Walter Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[79] Sven Schewe. Solving parity games in big steps. *FCTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, Springer LNCS 4855:449–460, 2007.

[80] Sven Schewe. Büchi Complementation Made Tight. *Symposium on Theoretical Aspects of Computer Science* (STACS 2009), pages 661-672, 2009.

[81] Sven Schewe. From parity and payoff games to linear programming. *Mathematical Foundations of Computer Science 2009*, Thirtyfourth International Symposium, MFCS 2009, Novy Smokovec, High Tatras, Slovakia, August 24-28, 2009. Proceedings. *Springer LNCS*, 5734:675–686, 2009.

[82] Thoralf Skolem. Begründung der elementaren Arithmetik durch die rekurrierende Denkweise ohne Anwendung scheinbarer Veränderlichen mit unendlichem Anwendungsbereich. *Videnskapsselskapets Skrifter I, Mathematisch-Naturwissenschaftliche Klasse* 6, 1923.

[83] Larry J. Stockmeyer. *Arithmetic versus Boolean operations in idealized register machines*, IBM Thomas J. Watson Research Center report RC 5954, 21 April 1976.

[84] Róbert Szelepcsényi. The method of forcing for nondeterministic automata. *Bulletin of the European Association for Theoretical Computer Science*, 96–100, 1987.

[85] Wai Yean Tan. *Reducibilities between regular languages.* Master Thesis, Department of Mathematics, National University of Singapore, 2010.

[86] Alfred Tarski. Der Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, 1:261–405, 1936.

[87] Axel Thue. Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln. *Norske Videnskabers Selskabs Skrifter, I, Mathematisch-Naturwissenschaftliche Klasse* (Kristiania), 10, 34 pages, 1914.

[88] Boris A. Trakhtenbrot. Turing computations with logarithmic delay. *Algebra i Logika*, 3:33-48, 1964.

[89] Alan M. Turing. On computable numbers with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936 and correction, 43:544–546, 1937.

[90] William M. Waite and Gerhard Goos. *Compiler Construction.* Texts and Monographs in Computer Science. Springer, Heidelberg, 1984.

[91] Daniel H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10:198–208, 1967.

[92] Jiangwei Zhang. *Regular and context-free languages and their closure properties with respect to specific many-one reductions.* Honours Year Project Thesis, Department of Mathematics, National University of Singapore, 2013.

[93] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200:135–183, 1998.