Dissertation

On the Relative Descriptional Complexity of Regular Expressions and Finite Automata

Stefan Gulan

Universität Trier

1. Gutachter: Prof. H. Fernau, Universität Trier
2. Gutachter: Prof. M. Holzer, Universität Gießen

Zusammenfassung

Reguläre Ausdrücke und endliche Automaten modellieren jeweils die Familie der regulären Sprachen. Derartige Sprachen bestehen aus Zeichenfolgen hoher Regelmäßigkeit und sind geeignet, um textuelle Muster, Kontrollflüsse sowie Systemspezifikationen formal – und damit algorithmisch – zu behandeln.

Es zeigt sich, dass Ausdrücke eine aus menschlicher Sicht intuitive Darstellung regulärer Sprachen erlauben. Andererseits lassen sich Automaten bereits als Datenstrukturen auffassen und bilden damit den Kern einer maschinellen Behandlung. Somit ergibt sich die die Frage nach einer effizienten Umwandlung zwischen den beiden Darstellungen. Für die Formalismen bestehen allerdings wesentliche Unterschiede hinsichtlich möglichen Kompaktheit der Darstellung, insbesondere bezüglich der jeweiligen Mindestgröße eines beschreibenden Objektes. Daraus ergeben sich wiederum theoretische Grenzen an die mögliche Effizienz der genannten Umwandlungen.

Derartige quantitative Aspekte bilden den wesentlichen Untersuchungsgegenstand der vorliegenden Arbeit. Sie gliedert sich in zwei Teile. Im ersten Teil werden zunächst Methoden zur Verkürzung von regulären Ausdrücken betrachtet und quantitativ bewertet. Weiter wird eine Konstruktion zur Umwandlung von Ausdrücken in Automaten vorgestellt, die ebendiese Verkürzung implizit vornimmt. Dabei zeigt sich, dass die angegebene Konstruktion das theoretisch bestmögliche Ergebnis bezüglich der relativen Größe von Ein- zu Ausgabe ergibt.

Im zweiten Teil der Arbeit werden die strukturellen Eigenschaften von Automaten ermittelt, die durch Ausdrücke nur mit erheblichem darstellerischem Mehraufwand beschrieben werden können. Dies geschieht indirekt, indem zunächst eine Klasse gerichteter Graphen untersucht und durch eine Menge abwesender Teilstrukturen charakterisiert wird. Es wird gezeigt, dass diese Klasse exakt diejenigen Strukturen enthält, die durch reguläre Ausdrücke kodiert werden können. Hieraus folgt, dass die abwesenden Teilstrukturen genau die Eigenschaften von Automaten darstellen, die besagten Mehraufwand verursachen.

Contents

1 Introduction

This work is concerned with two kinds of objects: regular expressions and finite automata. These formalisms describe regular languages, i.e., sets of strings that share a comparatively simple structure. Such languages — and, in turn, expressions and automata — are used in the description of textual patterns, workflow and dependence modeling, or formal verification. Testing words for membership in any given such language can be implemented using a fixed — i.e., finite — amount of memory, which is conveyed by the phrasing "finite" automaton. In this aspect they differ from more general classes, which require potentially unbound memory, but have the potential to model less regular, i.e., more involved, objects.

Other than expressions and automata, there are several further formalisms to describe regular languages (see, e.g. [57]). These formalisms are all equivalent and conversions among them are well-known. However, expressions and automata are arguably the notions which are used most frequently: regular expressions come natural to humans in order to express patterns, while finite automata translate immediately to efficient data structures. This raises the interest in methods to translate among the two notions efficiently. In particular, the direction from expressions to automata, or from human input to machine representation, is of great practical relevance.

Probably the most frequent application that involves regular expressions and finite automata is pattern matching in static text and streaming data. Common tools to locate instances of a pattern in a text are the `grep` application or its (many) derivatives, as well as `awk`, `sed` and `lex` (discussed extensively in [17]) Notice that these programs accept slightly more general patterns, namely "POSIX expressions". Concerning streaming data, regular expressions are nowadays used to specify filter rules in routing hardware; this topic is discussed in the introductory section of Ch. 5. These applications have in common that an input pattern is specified in form a regular expression while the execution applies a regular automaton.

As it turns out, the effort that is necessary to describe a regular language, i.e., the size of the descriptor, varies with the chosen representation. For example, in the case of regular expressions and finite automata, it rather easy to see that any regular expression can be converted to a finite automaton whose size is linear in that of the expression[1]. For the converse direction, however, it is known that there are regular languages for which the size of the smallest describing expression is exponential in the size of the smallest describing automaton.

This brings us to the subject at the core of the present work: we investigate conversions between expressions and automata and take a closer look at the properties that exert an influence on the relative sizes of these objects. We refer to the aspects involved with these consideration under the titular term of

Relative Descriptional Complexity.

The following section gives a brief survey over the particular aspects of this line of research that are addressed in the present work.

1.1 Per-Chapter Overview

Each chapter bears its own introduction, wherein the topic of the chapter is briefly introduced, motivated, and as far as possible accompanied by providing pointers to relevant related work. When fit, a chapter may be concluded with proposals for future research that might build on the findings presented in the chapter. The following overview states the main topic of each chapter and highlights the main results it contributes to the current state of research.

- In Ch. 2, we introduce the basic terminology and notation used throughout this document, mostly for well-known concepts, such as graphs and rewriting systems. Some fundamental results, on which many proofs will be based in this work are presented, notably Newman's Lemma and the principle of directional duality. Most of this chapter is a condensation of the definitions found standard textbooks, such as [1, 40, 27, 44]. While our terminology follows these references, and therefore usually the canonical nomenclature, the notation differs to some extents from that in the references, so it is advised to skim the chapter at least.

[1] To be precise, this is true only for so-called "nondeterministic" automata, see the introductory section of Ch. 3 for details.

- In Ch. 3 we consider the question how certain redundancies in regular expressions can be avoided. From an algorithmic point of view, we aim to remove these redundancies from expressions. This means that we start from an arbitrary expression and construct a second expression which is free of the redundant parts, while denoting the same language as the expression we started from. Our concern for efficiency dictates that such a construction operates on a purely syntactical level, meaning that the language denoted by either expression is not taken into account at all. We thus restrict ourselves to reorganizing the operators in an expression, which allows for some nontrivial adjustment nevertheless. This is realized by introducing two operators that rewrite an expression and terminate in a unique normal form, which is free of the redundancies under consideration. We find that if the input and output under this conversion differ at all, then the output — the normal form — is strictly smaller than the input expression. We further give an alternative characterization of expressions in this normal form by means of unary operators in the expression.

- The topic of Ch. 4 is the construction of finite automata from regular expressions. We present a rewriting system that operates on a class of automata. These automata generalize both expressions and finite automata, and allow for a lucid analysis of the construction's properties. For one, we show that the presented method produces unique outputs, i.e., it behaves as a function from expressions to automata. The construction is shown to be related to the normal form introduced in Ch. 3: we find that the generated automaton is invariant under taking this normal form of the input expression. We finally propose an implementations of the construction, taking several optimizing steps into account.

- In Ch. 5 we investigate in the quantitative aspects of the automaton-to-expression conversion presented in Ch. 4. In particular, we bound the size of an output automaton relative to the size of its input expression. To this end we infer an expression that maximizes this ratio; this expression will further be used for bounding the considered size ration from below, i.e., for any construction of this type. We will find that the upper bound for our construction coincides with the lower bound, which implies that the presented construction is optimal. This also settles the open question about how expression and automaton sizes relate fundamentally.

- In Ch. 6 we shift our focus towards purely graph theoretic considerations. Therein, we extend the well known class of arc-series-parallel digraphs. This class is defined as the graphs that can be constructed from an axiom graph

by repeated application of two rules, each of which replaces an arc with a slightly more complex subgraph. We augment this inductive definition by a third rule, one that allows to introduce loops. In doing so, we generalize the arc-series-parallel digraphs beyond the acyclic case. We proceed by showing that our new class can be efficiently decided, essentially by constructing the sequence of rule applications that may lead to a given graph backwards.

- In Ch. 7, we work out an alternative characterization of the class of graphs introduced in Ch. 6. This second characterization is of the obstruction set type, meaning that we give a set of structural properties — being graphs themselves — that are necessarily and sufficiently absent in graphs of our new class.

- Finally, in Ch. 8 we combine several of the results we found in the preceding parts of this work. Foremost, we present an encoding of the graphs we introduced in Ch. 6 by means of regular expressions. We then show that under some mild restrictions, this encoding in one-to-one and in a sense provides a bijection between regular expressions and such graphs. To this end we identify labeled graphs with the kind of automaton that is also considered in Ch. 4. The construction given in Ch. 4 will be used to decode expressions back to graphs. We also apply these findings to the construction of regular expressions from finite automata and, with help of the normal form presented in Ch. 3, provide a linear (upper) bound on the size of expression constructed from automata.

2 Basic Terminology and Notation

2.1 Sets, Relations, and Maps

The terms *set*, *class*, *family* and *collection* are used interchangeably. Enclosing braces for singleton sets may be omitted. The union of disjoint sets M and N is denoted $M \dot\cup N$.

Given sets M_i, $1 \leq i \leq n$, a set $R \subseteq M_1 \times M_2 \times \cdots \times M_n$ is a *relation* of *arity* n on the M_i. A relation of arity two is called a *binary* relation. A binary relation $R \subseteq M \times M$ is also called a relation *on* M. Members of a binary relation may be written infixed as xRy instead of $(x, y) \in R$.

The *product* of two binary relations $R \subseteq M_1 \times M_2$ and $Q \subseteq M_2 \times M_3$ is

$$RQ := \{(x, y) \mid \exists z : (x, z) \in R \wedge (z, y) \in Q\} \subseteq M_1 \times M_3.$$

The *dual* of a binary relation R is the binary relation

$$R^{-1} := \{(x, y) \mid (y, x) \in R\}$$

The n-*fold* iteration R^n of a binary relation R on M is defined inductively as $R^0 := \Delta M$ and $R^{n+1} := RR^n = R^n R$. The *transitive closure* R^+ and the *reflexive transitive closure* $R^\star$ of R are defined as

$$R^+ := \bigcup_{n \in \mathbb{N}^+} R^n \quad \text{resp.} \quad R^\star := \bigcup_{n \in \mathbb{N}} R^n$$

A *map* from A to B is a relation $h \subseteq A \times B$ s.t. for each $a \in A$ there is at most one $b \in B$ with $(a, b) \in h$. We write $h : A \to B$ to express that h is a map from A to B. Instead of $(a, b) \in h$ we also write $h(a) = b$ and say that h *maps* a to b. Further, we call b the *image* of a and a a *preimage* of b under h. An *injection* is a map $h : A \to B$ with at most one preimage of each $b \in B$ under h. If $h(a) = b$ for an injection h, we set $h^{-1}(b) := a$.

For $h : A \to B$ and $A_1 \subseteq A$, the *restriction* of h to A_1 is the map

$$h{\restriction}_{A_1} := h \cap (A_1 \times B).$$

2.2 Abstract Rewriting Systems

An *abstract rewriting system* (ARS) on M consists of a set M, called the *universe*, and a set of binary relations on M, called the *rules* of the ARS. We write

$$\mathfrak{A} = \langle M, \to_{\alpha_1}, \ldots, \to_{\alpha_n} \rangle$$

to denote the ARS $\mathfrak{A}$ with universe M and rules $\to_{\alpha_i} \subseteq M \times M$. A *sub-ARS* of the ARS $\langle M, \to_{\alpha_1}, \ldots, \to_{\alpha_n} \rangle$ is any ARS $\langle M', \to_{\beta_1}, \ldots, \to_{\beta_m} \rangle$ where $M' \subseteq M$ and each $\to_{\beta_i}$ is contained in some $\to_{\alpha_j}$.

An α_i-*rewriting* is any member of $\to_{\alpha_i}$, and an α_i-rewriting of m is any member of α_i with first element m, i.e., any $m \to_{\alpha_i} m'$. For $m \to_{\alpha_i} m'$ we might also refer to m' as an α_i-rewriting of m; it shall always be clear from the context which notion is used. An $\mathfrak{A}$-rewriting (of m) is any member of some $\to_{\alpha_i}$ (with m as first component).

Given a rewriting $m \to_{\alpha_i} m$, we say that applying $\to_{\alpha_i}$ to m *yields* m' or that m *rewrites* to m' with $\to_{\alpha_i}$. An $\mathfrak{A}$-rewriting of m may also be called a *rewriting step* in $\mathfrak{A}$. More generally, a *rewriting* of *length* k in $\mathfrak{A}$ is any sequence of rewritings $m \to_{\alpha_{i_1}} m_1$, $m_1 \to_{\alpha_{i_2}} m_2$, $\ldots$, $m_{k-1} \to_{\alpha_{i_{k-1}}} m_k$. A rewriting may be of length zero. More concisely, we write

$$m \to_{\alpha_{i_1}} m_1 \to_{\alpha_{i_2}} \cdots m_{k-1} \to_{\alpha_{i_{k-1}}} m_k, \text{ or just } m \to_{\alpha_{i_1}} \to_{\alpha_{i_2}} \cdots \to_{\alpha_{i_{k-1}}} m_k,$$

for a rewriting of length k. Notice that the second notation conveys less information then the first one, since the intermediate m_i are usually not uniquely determined.

If m admits no α_i-rewritings, it is α_i-*normal*. If m is α_i-normal for every rule $\to_{\alpha_i}$ of $\mathfrak{A}$, it is $\mathfrak{A}$-normal. If there is a rewriting from m to m' in $\mathfrak{A}$ s.t. m' is α_i-normal, then m' is an α_i-*normal form* of m in $\mathfrak{A}$. If m' is an α_i-normal form of m in $\mathfrak{A}$ for every rule in $\mathfrak{A}$, then m' is simply a normal form of m in $\mathfrak{A}$. The ARS $\mathfrak{A}$ is *terminating* if for every $m \in M$ there is some $k \in \mathbb{N}$ s.t. the length of no rewriting of m in $\mathfrak{A}$ exceeds k. This is equivalent to the property is that every exhaustive rewriting eventually leads to a normal form.

Two objects m, m' *converge* in $\mathfrak{A}$, denoted $m \sim_{\mathfrak{A}} m'$, if for some m'' there are rewritings $m \to^\star m''$ and $m' \to^\star m''$ in $\mathfrak{A}$.

An ARS $\langle M, R_1, \ldots, R_n \rangle$ is *locally confluent*, if for every divergent pair of rewritings mR_im' and mR_jm'' there is some m''' with rewritings $m'R_{i_1} \cdots R_{i_k}m'''$ and $m''R_{j_1} \cdots R_{j_l}m'''$ in this ARS. Local confluence is of utmost importance due to the following result by Newman:

Newman's Lemma ([39, 29]). An ARS that is terminating and locally confluent admits unique normal forms.

2.3 Graphs

A *finite directed pseudograph* is a 4-tuple $G = (V, A, t, h)$ where V and A are finite disjoint sets, called the *vertices*, resp. *arcs* of G, and t and h are maps from A to V, called the *tail*- and the *head*-maps of G. We will refer to such objects just as *graphs*. The capital letters F, G, and H, will always denote graphs. If G is not given explicitly, let $G = (V_G, A_G, t_G, h_G)$. The *order* of G is defined as $|V_G|$ while the *size* of G is $|A_G|$.

The vertices $t(a)$ and $h(a)$ are called the *tail* and the *head* of the arc a, respectively. An arc with head x and tail y is also called an arc *from* x *to* y, or just an xy-arc. We write $a = xy \in A$, if a is an xy-arc in A.[1] An xy-arc *leaves* x and *enters* y, is referred to as an *out-arc* of x and an *in-arc* of y. If G contains an xy-arc, x is called a *predecessor* of y in G, while y is called a *successor* of x in G. For $a = xy$, the vertices x and y are called the *endpoints* of a. We commonly refer to both $a = xy$ and $a' = yx$ as an *orientation* of an (otherwise unspecified) arc with endpoints x and y. An arc with coinciding endpoints is called a *loop*; more specifically, an arc $a = xx$ is called an x-loop. We say that x *carries* a loop, if an x-loop is present. Any arc that is not a loop is also called a *proper* arc.

The sets in-arcs and out-arcs of x in G are denoted $In_G(x)$ and $Out_G(x)$, respectively. The *in-degree* of a vertex x in G is the number of in-arcs of this vertex: $\mathrm{d}^-_G(x) = |In_G(x)|$; the *out-degree* of x is $\mathrm{d}^+_G(x) = |Out_G(x)|$. If G is known from the context, subscripts will be omitted. An xy-arc is a *constriction* if $\mathrm{d}^+(x) = \mathrm{d}^-(y) = 1$. A vertex x is *simple*, if $\mathrm{d}^-(x) = \mathrm{d}^+(x) = 1$.

Elementary operations on graphs

We are often interested in manipulating graphs, i.e., in the construction of new graphs from given ones. The manipulations considered in later chapters are combinations of a few elementary ones. In the following we consider graphs derived from G.

[1] The notation $a = xy$ is rather informal, since there might be multiple xy-arcs in a graph, i.e., xy does not determine a uniquely. The notation will only be used if it is irrelevant for the particular argument which particular xy-arc is considered.

- removing an arc a from G yields the graph
$$G \setminus a := (V_G,\, A_G \setminus a,\, t_G{\restriction}_{A\setminus a},\, h_G{\restriction}_{A\setminus a}).$$
- adding a new xy-arc a to G, assuming that $x, y \in A_G$, yields the graph
$$G \cup xy := (V_G,\, A_G \dot\cup a, t_G \cup (a,x), h_G \cup (a,y)).$$
- removing a vertex x from G yields the graph
$$G - x := (V_G \setminus x,\, A_G \setminus Inc(x),\, t_G{\restriction}_{A_G\setminus Inc(x)}, h_G{\restriction}_{A_G\setminus Inc(x)}).$$
- adding a new vertex x to G yields the graph
$$G + x := (V_G \dot\cup x,\, A_G,\, t_G, h_G).$$

In most cases, the order of two elementary operations that both manipulate arcs or that both manipulate vertices can be swapped without affecting the result. The following identities are inherited from the respective properties of set theoretic union and difference; formal proofs are omitted.

Proposition 2.3.1. *Let G be a graph.*

1. *For arbitrary x, y, the following identities hold*
   - $(G - x) - y = (G - y) - x$, $(G + x) + y = (G + y) + x$
   - $(G \setminus x) \setminus y = (G \setminus y) \setminus x$, $(G \cup x) \cup y = (G \cup y) \cup x$
2. *For distinct x, y, the following identities hold*
   - $(G - x) + y = (G + y) - x$, $(G \setminus x) \cup y = (G \cup y) \setminus x$

Based on Prop. 2.3.1, adding and removing vertices of arcs naturally extends to sets of arcs and vertices. For a set of arcs, $A = \{a_1, a_2, \ldots, a_n\}$, we thus write $G \setminus A$ instead of $(\cdots((G \setminus a_1) \setminus a_2)\cdots) \setminus a_n$. Likewise, we write $G \cup A$, and $G + V$ and $G - V$ for a set of vertices V.

A slightly less general property resembling Prop. 2.3.1 holds for the composition of a vertex operation with an arc operation. In one case we require that the vertex and arc that are manipulated are not incident. This is where we add an arc and then remove one of its incident vertices, which removes the arc, too. But swapping the order of these operations causes a "dangling" arc, which lacks a tail or a head. Nevertheless, we have the following:

Proposition 2.3.2. *Let G be a graph.*

1. *For arbitrary x, y, and z, the following identities hold*

- $(G - x) \setminus yz = (G \setminus yz) - x$
- $(G - x) \cup yz = (G \cup yz) - x$
- $(G + x) \setminus yz = (G \setminus yz) + x$

2. *For $x \notin \{y, z\}$ we find $(G + x) \cup yz = (G \cup yz) + x$*

To *merge* two vertices x and y in G yields the graph $G[x = y]$, which is informally described as the graph derived by identifying x and y, i.e., by replacing them with a "super-vertex" m, and redirecting all in-arcs and out-arcs of x and y to become in-arcs and out-arcs of m, respectively. The new vertex m is called the *merge vertex* of x and y in $G[x = y]$. The formal definition is

$$\begin{aligned} G[x = y] := G - \{x, y\} + m \cup \{zm \mid zx \in A_G \text{ or } zy \in A_G\} \\ \cup \{mz \mid zx \in A_G \text{ or } zy \in A_G\}. \end{aligned}$$

To *split* a vertex x in G yields the graph $G \ll x \gg$, which is informally described as the graph derived by replacing x with two vertices x_1, x_2, and an x_1x_2-arc, and redirecting all in-arcs of x to become in-arcs of x_1, and all out-arcs of x to become out-arcs of x_2. In particular, $G \ll x \gg$ contains an x_2x_1-arc for every x-loop in G. The vertices x_1 and x_2 are called the *split vertices* of x in $G \ll x \gg$. The formal definition is

$$\begin{aligned} G \ll x \gg := G - x + \{x_1, x_2\} \cup \{zx_1 \mid zx \in A_G,\, z \neq x\} \\ \cup \{x_2z \mid xz \in A_G,\, z \neq x\} \\ \cup \{x_2x_1 \mid xx \in A_G\}. \end{aligned}$$

In analogy to Props. 2.3.1 and 2.3.2, we establish some basic properties about swapping pairs of operations that involve merging and splitting. Extra care has to be taken for manipulation of arcs and vertices incident to these arcs, for vertices might be renamed. Again, these properties are stated in a matter-of-fact fashion; their proofs are routine.

Proposition 2.3.3. *Let x, y and z be distinct vertices in G.*

$$\begin{aligned} (G \cup xz)[x, y] &= G[x, y] \cup [x, y]z, & (G \cup zx)[x, y] &= G[x, y] \cup z[x, y] \\ (G \setminus xz)[x, y] &= G[x, y] \setminus [x, y]z, & (G \setminus xz)[x, y] &= G[x, y] \setminus [x, y]z \\ (G \cup xy) \ll x \gg &= G \ll x \gg \cup x'y, & (G \cup yz) \ll x \gg &= G \ll x \gg \cup yz \\ (G \setminus xy) \ll x \gg &= G \ll x \gg \setminus x'y, \end{aligned}$$

We call F a *subgraph* of G, denoted $F \subseteq G$, if there are sets $X \subseteq V_G$ and $Y \subseteq A_G$ s.t. $F = (G - X) \setminus Y$.

Walks, Paths and Cycles

An (x, y)-*walk* in G is a sequence of arcs $a_1 \dots a_n$, $a_i \in A_G$ s.t. $x = t(a_1)$, $h(a_i) = t(a_{i+1})$ for $1 \leq i < n$, and $h(a_n) = y$. Let $W = a_1 \dots a_n$ be an (x, y)-walk. As with arcs, W is said to leave x and enter y, and x and y are the endpoints of W. Every vertex z with $z = t(a_i)$ for $1 < i \leq n$, or $z = (h(a_i))$ for $1 \leq i < n$ is called an *internal vertex* of W. If x is an internal vertex of W, we say that W *passes through* x and that x *lies* on W. Observe that a vertex might well be both an endpoint and an inner vertex of a walk. An (x, y)-*segment* of a walk W is any subsequence of W that is an (x, y)-walk. If F_1 and F_2 are subgraphs of G, an (F_1, F_2)-walk in G is an (x, y)-walk where $x \in V_{F_1}$ and $y \in V_{F_2}$.

An (x, y)-*path* is an (x, y)-walk $a_1 \dots a_n$ where no vertex occurs as both endpoint and internal vertex, and for every internal vertex x there is exactly one $1 \leq i < n$ with $h(a_i) = x = t(a_{i+1})$. A path $P = a_1 \dots a_n$ in G is often identified as a subgraph of G; consequently, we denote the arcs that constitute a path P as A_P, and the vertices on this path as V_P. Two paths P_1 and P_2 are *internally disjoint*, if every vertex in $V_{P_1} \cap V_{P_2}$ is an endpoint of each P_i. A graph G is *strongly connected*, or just *strong*, if G contains an (x, y)-path for every distinct pair $x, y \in V_G$. All notions defined for walks carry over to paths.

A *cycle* in G is an (x, x)-path. A *chord* of the cycle $C \subseteq G$ is an arc $a \in A_G \setminus A_C$ whose endpoints lie on C. A graph is *acyclic* if it contains no cycles.

The *converse* of G is the graph $G^{\mathrm{R}} := (V_G, A_G, h_G, t_G)$, which is G with all arcs reversed. If G satisfies $G = G^{\mathrm{R}}$, it is called *self-converse*. An important concept with respect to the converse of a graph is the following meta-theorem:

Principle of Directional Duality. If G and H are in any graph-theoretic relation, then so are G^{R} and H^{R}.

2.4 Expressions and Automata

An *alphabet* is a nonempty set of symbols, called *letters*; the symbol $\mathcal{A}$, possibly indexed, always denotes an alphabet. The symbols ε and $\varnothing$ are assumed to not occur in any alphabet. A *word* over $\mathcal{A}$ is a finite sequence of letters from $\mathcal{A}$,

usually written by juxtaposition of letters. The empty sequence of letters is called the *empty word* and denoted ε, this word is defined over any alphabet.

The *concatenation* of two words w, w', denoted $w \cdot w'$ or simply ww', is the sequence consisting of the subsequences w and w' in that order. The concatenation of words is associative. The empty word ε is neutral wrt. this operation: for any word w holds $\varepsilon w = w\varepsilon = w$. The set of all words over $\mathcal{A}$ is denoted $\mathcal{A}^*$. A *language* L over $\mathcal{A}$ is a set of words over $\mathcal{A}$, i.e., $L \subseteq \mathcal{A}^*$. The concatenation of two languages L and L', denoted $L \cdot L'$, or again just LL', extends the concatenation of words naturally:

$$LL' := L \cdot L' := \{ww' \mid w \in L,\, w' \in L'\}.$$

The n-th *power* of a language L, denoted L^n, is defined recursively as

$$\begin{aligned} L^0 &:= \{\varepsilon\} \\ L^{k+1} &:= L^k L \end{aligned}$$

The *Kleene-closure* of L, denoted L^*, is defined as

$$L^* := \bigcup_{n \in \mathbb{N}} L^n$$

Regular Expressions A *regular expression* is a term that contains symbols from $\mathcal{A} \cup \{\varnothing, \varepsilon, \cdot, +, ^*, (,)\}$ for some $\mathcal{A}$ and adheres to a certain structure. A regular expression is usually referred to as just an *expression*. The set of expressions over $\mathcal{A}$, denoted $\mathrm{RE}_\mathcal{A}$ is defined as the smallest set that satisfies the following properties:

- $\varnothing \in \mathrm{RE}_\mathcal{A}$ and $\varepsilon \in \mathrm{RE}_\mathcal{A}$
- $a \in \mathrm{RE}_\mathcal{A}$ for every $a \in \mathcal{A}$
- If r and s are in $\mathrm{RE}_\mathcal{A}$, so are
  1. the *product* $(r \cdot s)$, where each of r and s is called a *factor*
  2. the *sum* $(r + s)$, where each of r and s is called an *addend*
  3. the *iteration* (r^*), where r is called the *base*
  4. the *option* $(r^?)$, where r is called the *choice*

The expressions $\varnothing$, ε, and a, for $a \in \mathcal{A}$, are referred to as *trivial* expressions or *literals*, while any other expression is also called a *compound* expression. In most cases, the particular alphabet over which an expression is defined, is irrelevant. We may thus write only $r \in \mathrm{RE}$ to state that r is a regular expression over some alphabet. Once more, the operator $\cdot$ is omitted and we write just rs as a shorthand for $r \cdot s$.

To each expression $r \in \mathrm{RE}_{\mathcal{A}}$ we associate a language $\mathrm{L}(r) \subseteq \mathcal{A}^*$, and we say that r *denotes* $\mathrm{L}(r)$. This language is defined recursively along the structure of r, as follows:

- $\mathrm{L}(\varnothing) := \emptyset$, $\mathrm{L}(\varepsilon) := \{\varepsilon\}$, and $\mathrm{L}(a) := \{a\}$ for $a \in \mathcal{A}$
- $\mathrm{L}((rs)) := \mathrm{L}(r)\,\mathrm{L}(s)$
- $\mathrm{L}((r + s)) := \mathrm{L}(r) \cup \mathrm{L}(s)$
- $\mathrm{L}((r^*)) := \mathrm{L}(r)^*$
- $\mathrm{L}((r^?)) := \mathrm{L}(r) \cup \varepsilon$

A language that is denoted by a regular expression is called a *regular language*.

Expressions following the above structure contain a plethora of parentheses, which affects readability. To allow for a more pleasing denotation we slightly loosen the syntax. Obviously, the outermost pair of parentheses can be omitted. The semantics of expressions suggest several further simplifications. For one, notice that the binary operators are associative. We thus omit parentheses for nested products and for nested sums. Moreover, we agree on the (decreasing) operator precedence ${}^{\mathrm{u}}, \cdot, +$, where ${}^{\mathrm{u}}$ is either of the unary operators. Parentheses between unary operators can be omitted as well, as their order of evaluation is unambiguous. For example, we write

$$((a(b(c^?))) + (d^*)) \quad \text{more concisely as} \quad abc^? + d^*.$$

Two expressions r and r' are *equivalent*, written $r \equiv r'$, if $\mathrm{L}(r) = \mathrm{L}(r')$. An expression r is *nullable* if $\varepsilon \in \mathrm{L}(r)$. The set of *subexpressions* of r, denoted $\mathrm{sub}(r)$, is defined recursively on r, as follows:

- $\mathrm{sub}(\varnothing) := \{\varnothing\}$, $\mathrm{sub}(\varepsilon) := \{\varepsilon\}$, and $\mathrm{sub}(a) := \{a\}$ for $a \in \mathcal{A}$;
- $\mathrm{sub}(rs) := \mathrm{sub}(r) \cup \mathrm{sub}(s) \cup \{rs\}$,
- $\mathrm{sub}(r + s) := \mathrm{sub}(r) \cup \mathrm{sub}(s) \cup \{r + s\}$,

- $\mathrm{sub}(r^*) := \mathrm{sub}(r) \cup \{r^*\}$
- $\mathrm{sub}(r^?) := \mathrm{sub}(r) \cup \{r^?\}$.

The set of *proper* subexpressions of r is defined as $\mathrm{sub}_{\neq}(r) := \mathrm{sub}(r) \setminus r$.

We use several notions to measure the frequency of symbols in an expression. The *alphabetic width* of r, denoted $|r|_{\mathcal{A}}$, is the number of literals occurring at different places in r. The formal definition is as follows:

- $|\varnothing|_{\mathcal{A}} := 1$, $|\varepsilon|_{\mathcal{A}} := 1$, and $|a|_{\mathcal{A}} := 1$, for $a \in \mathcal{A}$.
- $|st|_{\mathcal{A}} := |s|_{\mathcal{A}} + t$, $|s + t|_{\mathcal{A}} := |s|_{\mathcal{A}} + t$, $|s^*|_{\mathcal{A}} := |s|_{\mathcal{A}}$, and $|s^?|_{\mathcal{A}} := |s|_{\mathcal{A}}$.

We further write $|r|.$, $|r|_+$, $|r|_*$, and $|r|_?$ to denote the number products, sums, iterations, and options in r, respectively. We give the formal definition for $|r|_+$ only, the other definitions are derived by applying the obvious changes.

- $|\varepsilon|_+ = 0$, $|\varnothing|_+ = 0$, and $|a|_+ = 0$, for $a \in \mathcal{A}$.
- $|st|_+ = |s|_+ + |t|_+$, $|s+t|_+ = |s|_+ + |t|_+ + 1$, $|s^*|_+ = |s|_+$, and $|s^?|_+ = |s|_+$.

The *size* of r is now defined as

$$|r| := |r|_{\mathcal{A}} + |r|. + |r|_+ + |r|_* + |r|_?.$$

A *position* is a (possibly empty) sequence over $\{1, 2\}$. If σ is a position and r is an expression, then $(r)\langle\sigma\rangle$ denotes a subexpression of r, defined recursively on r and σ, as follows:

- $(r)\langle\rangle := r$
- $(r)\langle 1, \sigma'\rangle := \begin{cases} (s)\langle\sigma'\rangle & \text{for } r = st,\ r = s + t,\ r = s^*,\ \text{or } r = s^? \\ \text{undefined} & \text{otherwise.} \end{cases}$
- $(r)\langle 2, \sigma'\rangle := \begin{cases} (t)\langle\sigma'\rangle & \text{for } r = st,\ \text{ or } r = s + t \\ \text{undefined} & \text{otherwise.} \end{cases}$

We generally assume that $(r)\langle\sigma\rangle$ is defined. If $s \in \mathrm{sub}(r)$ and $(r)\langle\sigma\rangle = s$, we call σ an s-position in r. If s is a literal, a product, etc., we respectively call an s-position a literal position, a product position, etc., in r. It follows that $|r|_{\mathcal{A}}$ equals the number of literal positions in r, $|r|.$ equals the number of product positions, etc., and that $|r|$ equals the overall number of positions in r.

A *context*, denoted $r[\,]$ for some symbol r, is defined as follows:

- The *empty context* $[\,]$ is a context.
- If $r[]$ is a context and s is an expression, then $sr[\,]$, $r[\,]s$, and $s + r[\,]$ are contexts. Moreover, $(r[\,])^*$ and $(r[\,])^?$ are contexts.

A context $r[\,]$ should be understood as a regular expression with a "hole". Filling this hole with another expression s puts s in the context r. Let $r[s]$ denote the expression derived [2] from putting s in context r. For example, let $r[\,] = a + ((b[\,])^*c)^*$ and $s = ab^?$, then $r[s] = a + ((bab^?)^*c)^*$

Extended Finite Automata An *extended finite automaton* (*EFA*) over $\mathcal{A}$ is a quintuple $E = (Q, \mathcal{A}, \delta, I, F)$ of finite sets, as follows: Q is the set of *states*, $\mathcal{A}$ is an alphabet, $\delta \subset Q \times \mathrm{RE}_\mathcal{A} \times Q$ is called the *transition relation*, and $I \subseteq Q$ and $F \subseteq Q$ are the sets of *initial* resp. *final* states. We set $E = (Q_E, \mathcal{A}_E, \delta_E, I_E, F_E)$ if the components of E are not given explicitly. The class of EFAs over $\mathcal{A}$ is denoted $\mathbf{EFA}_\mathcal{A}$.

For E as above, the transition relation δ induces the relation $\vdash_E$ on $Q \times \mathcal{A}^*$, as follows: $(q, w_1w_2) \vdash_E (q', w_2)$ iff $(q, r, q') \in \delta$ and $w_1 \in \mathrm{L}(r)$. We say that the word w *carries* q to q' in E, if $(q, w) \vdash_E^\star (q', \varepsilon)$. This may also be written as $q \xrightarrow{w}_E q'$. The set of words that carry p to q in E is denoted $L_E(p, q)$, the formal definition is

$$L_E(p, q) := \{w \mid p \xrightarrow{w}_E q\}.$$

The language *accepted* by an EFA E, denoted $\mathrm{L}(E)$, is the set of all words that carry an initial state to a final one in E, so

$$\mathrm{L}(E) := \bigcup_{\substack{p \in I_E \\ q \in F_E}} L_E(p, q).$$

Two EFAs are *equivalent* if they accept the same language .

We further say that p *reaches* q in E, or equivalently, that q is *reached* from p, if some word carries p to q in E. Moreover, a state q is *accessible* in E if q is reached from an initial state. Symmetrically, p is *co-accessible*, if p reaches some final state. A state that is both accessible and co-accessible, is called *useful*. An EFA is *trim*, if all its states are useful.

[2] A formal definition of putting an expression in a context is easily given by induction on the structure of the context.

An EFA E is *normalized*, if it satisfies $I_E = \{q_0\}$, and $F_E = \{q_f\}$, s.t. $q_0 \neq q_f$ and there are no transitions (p, r, q_0) or (q_f, r, p) in E. We often assume that an EFA is trim and normalized. This is no serious restriction, as the following proposition shows.

Proposition 2.4.1. *For every EFA E there is an equivalent trim and normalized EFA E' s.t. $|Q_{E'}| \leq |Q_E| + 2$ and $|\delta_{E'}| \leq |\delta_E| + |I_E| + |F_E|$.*

Proof. Given an EFA E let Q_u denote the useful states in E, and let q_0 and q_f be two states that are not in Q_E. These states will be the unique initial and final states in E'. The transition relation of E is restricted to the set of useful states, and ε-transitions from q_0 to the initial states of E, resp. from the final states of E to q_f are added. Formally, let

$$\delta' := (\delta_E \cap Q_u \times \mathrm{RE}_{\mathcal{A}_E} \times Q_u) \cup \{(q_0, \varepsilon, q) \mid q \in I_E\} \cup \{(q, \varepsilon, q_f,) \mid q \in F_E\}.$$

Then the EFA $E' := (Q_u \cup \{q_0, q_f\}, \mathcal{A}_E, \delta', q_0, q_f)$ is normalized. The construction ensures that for $p \in I_E$ and $q \in F_E$, we have

$$p \xrightarrow{w}_{\delta_E} q \quad \text{iff} \quad q_0 \xrightarrow{\varepsilon}_{\delta'} p \xrightarrow{w}_{\delta'} q \xrightarrow{\varepsilon} q_f.$$

Since $w = \varepsilon w \varepsilon$, the EFAs E and E' are equivalent. The number of states and transitions of E' follows as in the claim. □

A *finite automaton* (FA) in the conventional sense is an EFA whose transition relation is restricted to $Q \times (\mathcal{A} \cup \{\varepsilon\}) \times Q$. The class of FAs over $\mathcal{A}$ is denoted $\mathbf{FA}_{\mathcal{A}}$. An FA is *deterministic* if its transition relation is further restricted to $Q \times \mathcal{A} \times Q$ and for $q \in Q$ and $a \in \mathcal{A}$ there is at most one transition (q, a, q').

It is often convenient to treat EFAs as graphs. To this end we define the *underlying graph* of an EFA E as the graph $\mathrm{G}(E) := (Q_E, \delta_E, \pi^3_1, \pi^3_3)$, where π^i_j denotes the projection of an i-tuple to its j-th component. The graph $\mathrm{G}(E)$ is also referred to as the *graph structure* of E.

We adopt graph theoretical terminology to EFAs, by attributing properties to states and transitions of E, which are actually properties of the vertices and arcs in $\mathrm{G}(E)$. We say that a state is "incident" to a transition, that a transition is a "loop transition", or that a state has "in-degree" n, and the like.

3 Simplifying Regular Expressions

Every regular language is denoted by an infinite number of distinct regular expressions. This becomes clear, e.g. by observing that for expressions r and s where $\mathrm{L}(s) \subseteq \mathrm{L}(r)$, the expressions r and $r + s$ are equivalent, so $\mathrm{L}(r)$ is denoted by the longer expressions $r+s$. We naturally prefer a concise expression, which we usually grasp more easily, not at least due to the fact that the very perception of the object takes less time. Considering computational aspects, shorter expressions are expected to allow for a more efficient processing, wrt. both memory consumption and running time. This is due to the fact that a first step in the computational utilization of expressions usually consists of converting an input expression into a finite automaton. In most cases, this automaton grows with the size of the expression.

There are exceptions to the argument above, primarily wrt. the correlation of expression and automaton sizes. First, if extended expressions — i.e., expressions with complement and intersection — are considered, an exponential blowup manifests in the initial construction of finite automata. An example, taken from Holzer & Kutrib [25], is the expression

$$r := (a+b)^* a \underbrace{(a+b)\cdots(a+b)}_{k \text{ times}} b(a+b)^*,$$

which allows for a finite automaton on $k+3$ states, as constructed from r by the method given in Ch. 4. However, the complement of r, containing a single additional operator, requires at least 2^{k-2} states in any equivalent automaton. Therefore although r and its complement are of "quite the same size", their equivalent automata differ significantly. However, as we shall not consider complement and intersection of expressions in this work, these matters will not concern us.

Second, there are constructions of finite automata from expressions that are non-monotonic, i.e, where shorter input expressions do not guarantee smaller output automata. This happens for constructions that realize on-the-fly optimizations, that implicitly remove some amount of redundancy from the output that is

present in the input. Therefore, a long redundant expression might be converted to a smaller automaton than a short non-redundant expression. To compare the sizes of the input and the output of such constructions in a meaningful way, the input expressions must be devoid of such redundancis; in other words, they must be normalized wrt. a particular property. Among these expressions the monotonic behavior is reinstated.

For example, consider the expressions $r = (a+b)^*$ and $s = a^*+a^*b(a+b)^*$, which denote the same language (example taken from [44]). Since the denoted language becomes obvious from r, but arguably not so much from s, a mechanical way to "simplify" s to r would come in handy. In particular, this leads to the following problems:

1. Decide whether two expressions are equivalent.
2. Shorten an expression by removing redundancies.

Observe that the questions whether two expressions denote the same language, and whether the language denoted by one expression is contained in that denoted by a second expressions are equivalent. This becomes evident from observing that for sets A, B, such as languages, we find 1) $A = B$ iff $A \subseteq B$ and $B \subseteq A$, and 2) $A \subseteq B$ iff $A \cup B = B$.

In some cases, the equivalence of two expressions is obvious from their structure. For example, the equivalence of the expressions $r + s$ and $s + r$ is immediately clear, without even considering languages denoted by r and s. Another example that does not rely on semantics would be the equivalence of r^* and r^{**}; this example also suggests a way to simplify expressions containing trivially nested iterations. These equivalences follow from algebraic laws that hold for regular operators wrt. the denoted languages: commutativity of sums and idempotency of iterations.

A further example that merely states an algebraic property is the equivalence of $rs + rt$ and $r(s + t)$. Again, this equivalence allows to shorten an expression based on its structure. Of course, this property generalizes in that $rs + r't$ can be simplified to the shorter $r(s + t)$ iff $r \equiv r'$ holds. The latter equivalence, however, is based not only on the structure of the longer expression, but also on the semantics of its subexpressions.

As far as the feasibility of these operations is concerned, it can be effectively decided whether two expressions denote the same language. The canonical procedure would be to construct the — uniquely determined — minimal equivalent deterministic finite automaton for each expression and then test these automata

for isomorphism. The catch is that there are expressions for which the size of any deterministic automaton is exponential in the size of the expression. For example, any deterministic FA that accepts the language denoted by

$$(a+b)^*a\underbrace{(a+b)(a+b)\ldots(a+b)}_{k \text{ times}}$$

requires more than 2^k states. Therefore, the running time of any method that involves the construction of deterministic automata is exponentially bound from below.

A seemingly different method to test expressions for equivalence is based on (nonlogical) deductive systems that formalize equational theories for expressions. Two complete systems of this type were given by Salomaa [45]. However, all efforts to employ such systems in a way that guarantees termination, come down to implicitly constructing the derivative automata of the inputs, which is a deterministic finite automaton again [19]. Therefore, the exponential lower bound holds for this approach, too.

Without blaming the inefficiency of deciding expression equivalence on the shortcomings of a particular method, it was shown by Meyer & Stockmeyer that the equivalence problem for regular expressions is PSPACE-hard [37]. This holds for deciding $r \equiv \mathcal{A}^*$ already. Hence, unless $\mathsf{P} = \mathsf{PSPACE}$, which is improbable, the size of expressions can not be reduced by semantic means in polynomial time.

On the positive side, a recent result by Hovland [28] shows that the equivalence of so-called one-unambiguous expressions [4, 9] can be decided in polynomial time. This is done by an axiomatic system that reduces either expression stepwise. The overall running time is $\mathcal{O}(n^4)$, where n equals the sum of alphabetic widths of the considered expressions. However, there are languages that do not allow for expressions of this type [9], meaning that the system is not complete.

3.1 Measures for Expressions and Languages

As yet, we were appealing to intuition about the size of an expressions, let us now consider a formalized notion. Various definitions for the size, or *complexity*, of regular expressions have been proposed; there is no general agreement in favor of a particular measure. The following measures — among others — are found in the literature:

1. The string length counts all symbols, including parentheses, occurring in the expression.
2. The reverse polish notation length counts the number of literals and regular operators occurring in an expression. The terminology comes from the fact that this measure coincides with the string length if an expression is written in reverse polish notation, and therefore contains no parentheses.
3. The alphabetic width, which counts the number of literals that occur in an expression. This equals the number of leaves in the parse of an expression.
4. The star height, defined as the maximal number of nested iterations.

The first systematic comparison of complexity measures for expression, including alphabetic width and star height, was done in the seventies by Ehrenfeucht & Zeiger [14, 15]. A more recent study by Ellul et al. [16] also takes string length and reverse polish notation length into consideration. Under some mild restrictions which prevent distortion of these measures, it is shown that for string length, reverse polish notation length and alphabetic width of an expression, each measure is at most a constant multiple of each other measure. In other words, these measures need not be distinguished for asymptotic analysis. There are still further complexity measures, each focusing on a different aspect of expression complexity. A quite exhaustive survey is provided by Holzer & Kutrib [26], who present various results about the interrelation of such measures in a unified framework.

As stated in the preliminaries, we settle for reverse polish notation length to measure for the complexity of expressions, and we refer to this value as the *size* of an expressions. To repeat the definition, the size of an expression r is defined as

$$|r| := |r|_{\mathcal{A}} + |r|_{\cdot} + |r|_{+} + |r|_{*} + |r|_{?}.$$

To motivate our choice of reverse polish notation for expression complexity, we briefly compare the size of an expression (as defined above) to string length and alphabetic width.

At the syntactic level, the difference between string length and size is merely a consequence of the infix notation we adopt for expressions; for reverse polish notation, i.e., postfix notation, the two notions coincide. [1]

[1] String length, of course, comes with other merits. Consider an expression that contains only parentheses that can not be removed due to associativity and precedence of operators,

The alphabetic width contains no information at all about the number of unary operators in an expression. This does not serve our purpose — which is to simplify expressions by possibly reducing the number of unary operators; the effects of such a simplification on expression complexity are not reflected by this measure. Nevertheless, we will use alphabetic width of expressions for evaluating the conciseness of an expression in terms of the number of unary operators. In particular, we will show that the number of unary operators can be bounded by the alphabetic width in several ways.

The star height is a rather interesting measure, although one that puts exclusive focus on a very specific aspect of expression complexity. Rather than formalizing our intuition of expression "conciseness", this measure conveys information about a particular graph-structural property of equivalent automata: its "cycle rank". Also, the star height of an expression does not relate linearly to the measures discussed above. To see this, observe that any expression that denotes a finite language has star height zero — that is, if we assume that subexpressions such as s^*, for $s \equiv \varnothing$ or $s \equiv \varepsilon$, are absent. On the other hand, a finite language may be of arbitrary cardinality, which is reflected in the respective expression. Further (nontrivial) results are given in [26].

Following the discussion in this chapter's introductory section, it seems unlikely that the alphabetic width of arbitrary expressions can be reduced efficiently, as this would require to analyze the equivalence of subexpressions. This consequently holds for the number of binary operators, too, which is related to alphabetic width by

$$|r|_{\mathcal{A}} = |r|_{\cdot} + |r|_{+} + 1.$$

This leaves the number of unary operators as the only contributor to expression size that can possibly be reduced efficiently. We set

$$|r|_{\omega} := |r|_{*} + |r|_{?}$$

to denote the *unary width* of an expression r.

In order to measure the conciseness of an expression wrt. its unary width in a meaningful way, some caution is still required. The unary width alone does not reflect this intuition properly, since it might be possible to "trade" unary operators for longer expressions. For example, the expression $a^?b^?$ is equivalent to $\varepsilon + a + b + ab$, whose unary width is certainly minimal, yet the overall

or are superfluous (which our expression syntax does not allow anyway). For such an expressions, the string length, compared to the size, contains information on the number of sums in products, as in $r(s + t)$ and further on nontrivial bases, such as $(st)^*$.

expression is less concise. We therefore need to take every equivalent expression into account, which comes down to considering the denoted language. This motivates the following

Definition 1. Let L be any regular language. The *alphabetic complexity* of L is defined as the least alphabetic width among all expressions denoting L:

$$\alpha(L) := \min\{|r|_{\mathcal{A}} \mid \mathrm{L}(r) = L\}.$$

The *relative unary complexity* of L is defined as the least unary width among all expressions denoting L relative to the alphabetic complexity of L:

$$\omega(L) := \min\{\frac{|r|_{\omega}}{\alpha(L)} \mid \mathrm{L}(r) = L\}.$$

In the following, we frequently use the following equivalences that are easily verified from the semantics of iteration and option:

- Idempotency of unary operators: $r^{**} \equiv r^{*}$ and $r^{??} \equiv r^{?}$.
- Absorption by star: $r^{*?} \equiv r^{*}$ and $r^{?*} \equiv r^{*}$.

These equivalences can be put to immediate use for the simplification of expressions by syntactic means. This yields a first upper bound on the unary complexity of languages.

Proposition 3.1.1. *Let L be a regular language. Then the unary complexity of L is bounded from above, as follows:*

$$\omega(L) \leq 2 - \frac{1}{|\Sigma_L|}.$$

Proof. Let r be any expression s.t. $|r|_{\mathcal{A}} = \alpha(L)$. We construct an expression equivalent to r by using idempotency of the unary operators and absorption by star. That is, we replace every subexpression s^{**} or $s^{??}$ of r with s^{*}, resp. s^{*}. We further replace every subexpression $s^{*?}$ or $s^{?*}$ with s^{*}. Each such replacement yields an equivalent expression. Iterating this process terminates in an expression r' which does not contain stacked unary operators. Therefore, every operand of a unary operator is a literal, a product, or a sum. Thus r' satisfies

$$|r'|_{\omega} \leq |r'|_{\mathcal{A}} + |r'|_{\cdot} + |r'|_{+} = 2|r'|_{\mathcal{A}} - 1.$$

Observe that the replacements we apply leave the literals of the original expression "untouched", so we find $|r'|_{\mathcal{A}} = |r|_{\mathcal{A}}$. Since we assumed $|r|_{\mathcal{A}} = \alpha(L)$, and by the definition of alphabetic complexity, we find

$$\omega(L) \leq \frac{|r'|_\omega}{\alpha(L)} = \frac{2|r|_{\mathcal{A}} - 1}{|r|_{\mathcal{A}}} = 2 - \frac{1}{|r|_{\mathcal{A}}} \leq 2 - \frac{1}{|\Sigma_L|}.$$

□

The expression r', as constructed in Prop. 3.1.1, can be derived from r in time $\mathcal{O}(|r|)$. A straightforward implementation consists of a term rewriting system[2] that replaces a left-hand side of an equivalence in the proof of above proposition with its corresponding right-hand side. It is also easy to see that r' is unique. We will not pursue this construction further; instead, in the next section, we present a more powerful method which subsumes the one just described and improves on the bound on $\omega(L)$. These results were in part published in [21].

3.2 Strong Star Normal Form

The original star normal form of regular expressions was introduced by Brüggemann-Klein in order to speed up the construction of position-automata from expressions from cubic to quadratic time [8]. An expression in star normal form is devoid of a single redundancy, namely, the nullability of bases. In other words, if r is in star normal form and $s^* \in \mathrm{sub}(r)$ then $\varepsilon \notin \mathrm{L}(s)$.

In this section we generalize the star normal form to handle expressions with options, i.e., the operator $^?$. The resulting normal form is shown to subsume the original one. We keep close to Brüggemann-Klein's notation, using two operators which act as maps on expression, resp. subexpressions.

Definition 2. The operators $^\circ$ and $^\bullet$ are recursively defined on regular expressions as follows.

- $[\cdot]^\circ$: $\varepsilon^\circ := \varnothing^\circ := \varnothing$, $a^\circ := a$ for $a \in \mathcal{A}$, $r^{?\circ} := r^\circ$, $(r+s)^\circ := r^\circ + s^\circ$, $r^{*\circ} := r^\circ$ and
$$(rs)^\circ := \begin{cases} r^\circ + s^\circ & \text{if } \varepsilon \in \mathrm{L}(rs) \\ rs & \text{else} \end{cases}$$

[2] We do not care about the actual implementation of term rewriting systems (for details, see, e.g., [40]). It suffices to know that each single rewriting step requires time $\mathcal{O}(1)$.

- $[\cdot]^\bullet$: $\varnothing^\bullet := \varnothing$, $\varepsilon^\bullet := \varepsilon$, $a^\bullet := a$ for $a \in \mathcal{A}$, $(rs)^\bullet := r^\bullet s^\bullet$, $(r+s)^\bullet := r^\bullet + s^\bullet$, $r^{*\bullet} := r^{\circ\bullet *}$ and
$$r^{?\bullet} := \begin{cases} r^\bullet & \text{if } \varepsilon \in \mathrm{L}(r) \\ r^{\bullet ?} & \text{else} \end{cases}$$

The expression $r^\bullet$ is called the *strong star normal form* (SSNF) of r.

A contribution of the presented variation in itself is that the definition of the operator $^\circ$ is simplified in comparison to the original work. Therein, $(st)^\circ$ is overdetermined: instead of the necessary two cases, as above, four cases are used. Since the strong star normal form coincides with the traditional one if option-free expressions are considered, this improves over the original formulation, notably easening inductive proofs.

One might argue that the distinction of cases for $(st)^\circ$ and $s^{?\bullet}$ by nullability of the operand is not a syntactic criterion but a semantic one. This, however, is merely a matter of notation — the nullable expressions can alternatively be defined syntactically; for convenience we use the Backus-Naur form. Recall that $\mathrm{RE}_\mathcal{A}$ denotes the class of regular expressions over $\mathcal{A}$. The proper subset $\mathrm{nRE}_\mathcal{A}$ of nullable regular expressions over $\mathcal{A}$ is given by

$$\mathrm{nRE} ::= \varepsilon \mid \mathrm{nRE} \cdot \mathrm{nRE} \mid \mathrm{RE} + \mathrm{nRE} \mid \mathrm{nRE} + \mathrm{RE} \mid \mathrm{RE}^* \mid \mathrm{RE}^? \,.$$

Formally, the operators $^\circ$ and $^\bullet$ define maps $\mathrm{RE} \to \mathrm{RE}$, thus the strong star normal form of an expression, i.e., the image of r under $^\bullet$, is certainly unique.

Example. We compute the SSNF of $r = ((a^*b^?) + a + c^?)^*$. Steps that do not depend on another will be performed in parallel, and for $a \in \Sigma$, $\mathrm{op} \in \{*, ?\}$, we perform two steps at once by writing $a^{\mathrm{op}\circ} = a$.

$$\begin{aligned} r^\bullet &= ((a^*b^?) + a + c^?)^{*\bullet} = ((a^*b^?) + a + c^?)^{\circ\bullet *} \\ &= ((a^*b^?)^\circ + a^\circ + c^{?\circ})^{\bullet *} = ((a^{*\circ} + b^{?\circ}) + a + c)^{\bullet *} \\ &= (a + b + a + c)^{\bullet *} = (a^\bullet + b^\bullet + a^\bullet + c^\bullet)^* \\ &= (a + b + a + c)^* \end{aligned}$$

An important property of SSNF, i.e., the operator $^\bullet$, is that expressions are not mapped to longer ones. This actually holds for either operator that is involved in computing the SSNF. Regarding our initial motivation — the simplification of expressions — this is certainly relevant. The property might be clear from the definitions of $^\circ$ and $^\bullet$ already, where no regular operator is added. For the sake

of being formally sound, we prove this claim by induction on the expressions structure.

Proposition 3.2.1 (Monotonicity)**.** *For any expression r we find*

$$|r^\circ| \leq |r| \quad \textit{and} \quad |r^\bullet| \leq |r|.$$

Proof. We prove the claim for the operator $^\circ$ first. The claim is true for the three base cases, which satisfy $|\varnothing^\circ| \leq |\varnothing|$, $|\varepsilon^\circ| \leq |\varnothing|$, and $|a^\circ| \leq |a|$ for $a \in \mathcal{A}$. Assume the claim is true for expressions r and s. For rs, the normal form $(rs)^\circ$ is either $r^\circ + s^\circ$ or rs, depending on the nullability of rs. In the latter case, the claim follows immediately, whereas in the former case, it follows due to

$$|(rs)^\circ| = |r^\circ s^\circ| = |r^\circ| + |s^\circ| + 1 \overset{I_A}{\leq} |r| + |s| + 1 = |rs|.$$

By analogous reasoning, we find that the claim holds for $s + t$ as well. For $r^?$ we find

$$|r^{?\circ}| = |r^\circ| \overset{I_A}{\leq} |r| < |r^?|,$$

and for r^* the analogous argument provides the claim again.

We can now prove the claim for $^\bullet$, too. For trivial expressions, we find $|\varnothing^\bullet| \leq |\varnothing|$, $|\varepsilon^\bullet| \leq |\varepsilon|$, and $|a^\bullet| \leq |a|$ for $a \in \mathcal{A}$. Let the claim be true for r and s, and consider the compound expressions. For a product, we have

$$|(rs)^\bullet| = |r^\bullet s^\bullet| = |r^\bullet| + |s^\bullet| + 1 \overset{I_A}{\leq} |r| + |s| + 1 = |rs|;$$

again, the inductive step is the same for the sum $r + s$. For the iteration r^*, we use that we have already shown monotonicity of the operator $^\circ$. Thus follows

$$|r^{*\bullet}| = |r^{\circ\bullet *}| \leq |r^{\bullet *}| = |r^\bullet| + 1 \overset{I_A}{\leq} |r| + 1 = |r^*|.$$

Finally, for an option $r^?$ we need to consider two cases again, depending on the nullability of the operand. In case that r is nullable, we have $r^{?\bullet} = r^\bullet$, i.e., the operator is simply removed and the claim follows from the inductive assumption. Otherwise, we obtain

$$|r^{?\bullet}| = |r^{\bullet ?}| = |r^\bullet| + 1 \overset{I_A}{\leq} |r| + 1 = |r^?|.$$

This completes the proof. □

Apart from showing that SSNF does at least not behave contrary to our primary intention, Prop. 3.2.1 also allows us to prove further properties of $^\circ$ and $^\bullet$ by induction over the length of expressions. In fact, structural induction is not always applicable, as will become apparent in the proof of Prop. 3.2.3.

To prove further properties of our normal form, or equivalently, the operator $^\bullet$, we need to examine the "inner" operator $^\circ$ first. To this end, we make use of two equivalences involving iterations.

Proposition 3.2.2.

1. *Any pair of expressions r, s, satisfies*

$$(r + s)^* \equiv (r^* + s^*)^*. \tag{3.1}$$

2. *Any pair of nullable expressions r, s, satisfies*

$$(rs)^* \equiv (r + s)^*. \tag{3.2}$$

Proof. Either statement is proven by mutual inclusion of the languages denoted by the left and right hand sides.

1. Let $w \in \mathrm{L}((r+s)^*)$, then $w = w_1 \dots w_n$, s.t. $w_i \in \mathrm{L}(r)$ or $w_i \in \mathrm{L}(s)$. Then $w_i \in \mathrm{L}(r^*)$ or $w_i \in \mathrm{L}(s^*)$, which yields $w \in \mathrm{L}((r^* + s^*)^*)$.

   For $w \in \mathrm{L}((r^*+s^*)^*)$, we have $w = w_1 \dots w_n$ s.t. $w_i \in \mathrm{L}(r^*)$ or $w_i \in \mathrm{L}(s^*)$. Thus, for each i, $w_i = v_{i,1} \dots v_{i,k_i}$ with $v_{i,j} \in \mathrm{L}(r)$ or $v_{i,j} \in \mathrm{L}(s)$. Therefore $w \in \mathrm{L}((r + s)^*)$.

2. Let $w \in \mathrm{L}((rs)^*)$, then $w = w_1 \dots w_n$ s.t. $w_i \in \mathrm{L}(r)$ for odd i and $w_i \in \mathrm{L}(s)$ for even i. In particular, $w_i \in \mathrm{L}(r + s)$, thus $w \in \mathrm{L}((r + s)^*)$.

   Since r and s are both nullable, $r \equiv r + \varepsilon$ and $s \equiv s + \varepsilon$ hold. Thus rs is equivalent to $(r+\varepsilon)(s+\varepsilon)$, and we use distributivity to find $(r+\varepsilon)(s+\varepsilon) \equiv rs+r\varepsilon+\varepsilon s+\varepsilon$, which simplifies to $rs+r+s$. Now $\mathrm{L}(r+s) \subseteq \mathrm{L}(rs+r+s)$ obviously holds, thus also $\mathrm{L}((r + s)^*) \subseteq \mathrm{L}((rs + r + s)^*) = \mathrm{L}((rs)^*)$.

□

In a sense, the second equivalence stated in Prop. 3.2.2 is "hard-coded" into the definition of SSNF, which sets $(st)^\circ = s^\circ + t^\circ$ for nullable st. With this, we are set to investigate further properties of the operator $^\circ$.

Proposition 3.2.3. *For any expression r we find*

1. $\varepsilon \notin \mathrm{L}(r^\circ)$
2. $r^{\circ\circ} = r^\circ$
3. $r^{\circ *} \equiv r^*$

Proof.

1. For trivial expressions, $r \in \mathcal{A} \cup \{\varepsilon, \varnothing\}$, the claim holds due to $a^\circ = a$, $\varepsilon^\circ = \varnothing$ and $\varnothing^\circ = \varnothing$. Assume the claim holds for any expression smaller than r. For $r = s + t$ the inductive step is straightforward since $(s+t)^\circ \stackrel{\text{def}}{=} s^\circ + t^\circ$ where both summands are non-nullable by assumption. For $r = st$ with $\varepsilon \notin \mathrm{L}(st)$, the claim is trivial. Otherwise, if $\varepsilon \in \mathrm{L}(st)$, we have $(st)^\circ \stackrel{\text{def}}{=} s^\circ + t^\circ$, and the inductive assumption applies to both s and t. Thus, either case yields $\varepsilon \notin \mathrm{L}(st^\circ)$. For $r = s^*$ we have $s^{*\circ} \stackrel{\text{def}}{=} s^\circ$. The inductive assumption applies to s, so $r^\circ = s^\circ$ implies $\varepsilon \notin \mathrm{L}(r^\circ)$. An analogous argument provides the claim for $r = s^?$.

2. Again, the claim holds for $r \in \mathcal{A} \cup \{\varepsilon, \varnothing\}$, so assume it holds for any expression smaller than r. For a sum $r = s + t$ we get

$$(s+t)^{\circ\circ} \stackrel{\text{def}}{=} (s^\circ + t^\circ)^\circ \stackrel{\text{def}}{=} s^{\circ\circ} + t^{\circ\circ} \stackrel{I_A}{=} s^\circ + t^\circ \stackrel{\text{def}}{=} (s+t)^\circ.$$

   For a non-nullable product $r = st$, we get $(st)^{\circ\circ} = (st)^\circ = st$ immediately from the definition. A nullable product on the other hand yields

$$(st)^{\circ\circ} \stackrel{\text{def}}{=} (s^\circ + t^\circ)^\circ \stackrel{\text{def}}{=} s^{\circ\circ} + t^{\circ\circ} \stackrel{I_A}{=} s^\circ + t^\circ \stackrel{\text{def}}{=} (st)^\circ,$$

   where the first and the last step make use of nullability. For $r = s^*$, and, by analogy, for $r = s^?$, we get

$$s^{*\circ\circ} \stackrel{\text{def}}{=} s^{\circ\circ} \stackrel{I_A}{=} s^\circ \stackrel{\text{def}}{=} s^{*\circ}.$$

3. The claim is true if r is a letter, ε, or $\varnothing$. Assume it is true for expressions smaller than r. In case $r = s + t$ we use the equivalence (3.1) to proceed with

$$(s+t)^{\circ *} \stackrel{\text{def}}{=} (s^\circ + t^\circ)^* \stackrel{(3.1)}{\equiv} (s^{\circ *} + t^{\circ *})^* \stackrel{I_A}{\equiv} (s+t)^*,$$

   where the last step uses $|s^\circ| \leq |s|$, proven in Prop. 3.2.1, to the effect that the inductive assumption applies. For the product $r = st$ we need to distinguish again whether st is nullable. If it is not, the claim follows as a

trivial consequence of $(st)^\circ \overset{\text{def}}{=} st$. If st is nullable, we use the equivalences stated in Prop. 3.2.2 once again:

$$(st)^{\circ *} \overset{\text{def}}{=} (s^\circ + t^\circ)^* \overset{(3.1)}{\equiv} (s^{\circ *} + t^{\circ *})^* \overset{I_A}{\equiv} (s^* + t^*)^* \overset{(3.1)}{\equiv} (s+t)^* \overset{(3.2)}{\equiv} (st)^*$$

In the case of an iteration, $r = s^*$, the idempotency of iteration yields:

$$s^{*\circ *} \overset{\text{def}}{=} s^{\circ *} \overset{I_A}{\equiv} s^* \equiv s^{**}.$$

For the final step, consider the option $r = s^?$. We use absorption by star in the last step of proving the claim for this case, as follows:

$$s^{?\circ *} \overset{\text{def}}{=} s^{\circ *} \overset{I_A}{\equiv} s^* \equiv s^{?*}.$$

□

With the third item of Prop. 3.2.3 we can now show that the language denoted by an expression is invariant under taking the strong star normal form.

Lemma 3.2.4 (Invariance)**.** *For any expression r we find $r^\bullet \equiv r$.*

Proof. The claim trivially holds for expressions of size 1, so assume it holds for any expression smaller than r. For $r = st$ we find

$$(st)^\bullet \overset{\text{def}}{=} s^\bullet t^\bullet \overset{I_A}{\equiv} st,$$

and similarly for $r = s + t$:

$$(s+t)^\bullet \overset{\text{def}}{=} s^\bullet + t^\bullet \overset{I_A}{\equiv} s + t.$$

For $r = s^*$, we use Prop. 3.2.3, which yields

$$s^{*\bullet} \overset{\text{def}}{=} s^{\circ \bullet *} \equiv s^{*\bullet} \overset{I_A}{\equiv} s^*.$$

Finally for $r = s^?$ assume first that s is nullable. Then the option is redundant, i.e., the equivalence $s^? \equiv s$ holds, hence

$$s^{?\bullet} \equiv s^\bullet \overset{I_A}{\equiv} s \equiv s^?.$$

If otherwise s is not nullable, the inductive step is

$$s^{?\bullet} \overset{\text{def}}{=} s^{\bullet ?} \overset{I_A}{\equiv} s^?,$$

which completes the proof. □

We need the auxiliary fact that the operators $^\circ$ and $^\bullet$ commute, i.e., that they can be applied to an expression in either order, leading to the same result.

Proposition 3.2.5. *For any expression r we find $r^{\circ\bullet} = r^{\bullet\circ}$.*

Proof. For $r \in \{\varepsilon, \varnothing\}$, the definition of SSNF yields $r^{\circ\bullet} = r^{\bullet\circ} = \varnothing$. Likewise, for a letter a we find $a^{\circ\bullet} = a^{\bullet\circ} = a$ immediately. Assume that the claim is true for expressions smaller than r, and observe that Lem. 3.2.4 implies that $r^\bullet$ is nullable iff r is nullable.

If r is a product, $r = st$, distinguish whether st is nullable. If it is, we find

$$(st)^{\circ\bullet} \overset{\text{def}}{=} (s^\circ + t^\circ)^\bullet \overset{\text{def}}{=} s^{\circ\bullet} + t^{\circ\bullet} \overset{I_A}{=} s^{\bullet\circ} + t^{\bullet\circ} \overset{\text{def}}{=} (s^\bullet t^\bullet)^\circ \overset{\text{def}}{=} (st)^{\bullet\circ}.$$

Otherwise, if st is not nullable, then neither is $(st)^\bullet$. By definition of the operator $^\circ$, we have $(st)^\circ = st$ and $(st)^{\bullet\circ} = (st)^\bullet$ in that case. This yields

$$(st)^{\circ\bullet} \overset{\text{def}}{=} (st)^\bullet \overset{\text{def}}{=} (st)^{\bullet\circ}.$$

The case $r = s + t$ is straightforward, and thus slightly abridged:

$$(s+t)^{\circ\bullet} \overset{\text{def}}{=} s^{\circ\bullet} + t^{\circ\bullet} \overset{I_A}{=} s^{\bullet\circ} + t^{\bullet\circ} \overset{\text{def}}{=} (s+t)^{\bullet\circ}.$$

For an iteration, $r = s^*$, we use that the operator $^\circ$ is idempotent, as shown in Prop. 3.2.3. With this we find

$$s^{*\circ\bullet} \overset{\text{def}}{=} s^{\circ\bullet} \overset{I_A}{=} s^{\bullet\circ} = s^{\bullet\circ\circ} \overset{\text{def}}{=} s^{\bullet\circ *\circ} \overset{\text{def}}{=} s^{*\bullet\circ}$$

The option $r = s^?$ again has two subcases. If s is nullable, we find

$$s^{?\circ\bullet} \overset{\text{def}}{=} s^{\circ\bullet} \overset{I_A}{=} s^{\bullet\circ} \overset{\text{def}}{=} s^{?\bullet\circ},$$

whereas otherwise, the inductive step is as follows:

$$s^{?\circ\bullet} \overset{\text{def}}{=} s^{\circ\bullet} \overset{I_A}{=} s^{\bullet\circ} \overset{\text{def}}{=} s^{\bullet ?\circ} \overset{\text{def}}{=} s^{?\bullet\circ}.$$

Thus the claim holds for any expression, and the statement follows. □

With the aid of Prop. 3.2.5 we show that $^\bullet$ is idempotent, too.

Lemma 3.2.6 (Idempotency)**.** *For any expression r we find $r^{\bullet\bullet} = r^\bullet$.*

Proof. The claim is certainly true for r being $\varnothing$, ε, or a letter, so assume it is true for expressions smaller than r. For r being a product, $r = st$, the inductive step is straightforward:

$$(st)^{\bullet\bullet} = (s^\bullet t^\bullet)^\bullet = s^{\bullet\bullet} t^{\bullet\bullet} \overset{I_A}{=} s^\bullet t^\bullet = (st)^\bullet.$$

The inductive steps for sums is completely analogous to that for products. For an iteration $r = s^*$ we use Props. 3.2.3 and 3.2.5:

$$s^{*\bullet\bullet} = s^{\circ\bullet *\bullet} = s^{\circ\bullet\circ\bullet *} = s^{\circ\circ\bullet\bullet *} = s^{\circ\bullet\bullet *} \overset{I_A}{=} s^{\circ\bullet *} = s^{*\bullet}.$$

For $r = s^?$, the inductive step is straightforward if s is not nullable:

$$s^{?\bullet\bullet} = s^{\bullet ?\bullet} = s^{\bullet\bullet ?} \overset{I_A}{=} s^{\bullet ?} = s^{?\bullet}.$$

On the other hand, if s is nullable, we have

$$s^{?\bullet\bullet} = s^{\bullet\bullet} \overset{I_A}{=} s^\bullet = s^{?\bullet},$$

and the proof is complete. □

Since the operator $^\bullet$ is idempotent, realizes a map on expressions, and does not alter the language denoted by an expression, it is justified to call $r^\bullet$ a "normal form" of r. We say that r is *in strong star normal form*, resp., that r is in SSNF, if $r = r^\bullet$.

It is not immediately obvious from the recursive definition of SSNF by $^\circ$ and $^\bullet$ which properties are satisfied by an expression in SSNF. It should be clear that some redundancies must be removed if the SSNF of an expression is shorter than the expressions itself, since the denoted languages are equal. We will show that these redundancies are options and iterations whose main operand is nullable, i.e., any subexpression $s^?$ and s^* where $\varepsilon \in \mathrm{L}(s)$.

To prove this claimed characterization of expressions in SSNF, we first need to show a preliminary result concerning the operator $^\circ$.

Proposition 3.2.7. *For any expression r we find $r^\circ = r$ iff $\varepsilon \notin \mathrm{L}(r)$.*

Proof. In Prop. 3.2.3 we have shown that $\varepsilon \notin \mathrm{L}(r^\circ)$ holds for arbitrary r. Therefore, $r^\circ = r$ implies $\varepsilon \notin \mathrm{L}(r)$.

The converse direction is proven by induction on the length of non-nullable expressions. The base cases are $r = \varnothing$ and $r = a \in \mathcal{A}$, where $r^\circ = r$ holds by

definition. Let r be non-nullable and assume that the claim holds for expressions smaller than r.

If $r = st$, then

$$r^\circ = (st)^\circ \stackrel{\text{def}}{=} st = r,$$

since we assumed that r is not nullable. For $r = s + t$, the assumption that r is not nullable implies that neither s nor t is nullable, so the inductive assumption applies to both. We find

$$r^\circ = (s + t)^\circ \stackrel{\text{def}}{=} s^\circ + t^\circ \stackrel{I_A}{=} s + t = r.$$

For r being an option or an iteration, the claim follows trivially since r is nullable in either case. □

Theorem 3.2.8. *The following statements are equivalent:*

1. $r = r^\bullet$, *i.e., r is in* SSNF
2. $\forall s^*, s^? \in \mathrm{sub}(r) : \varepsilon \notin \mathrm{L}(s)$

Proof.

- 1. $\Rightarrow$ 2.
  We show by induction over the size of r that $s^*, s^? \in \mathrm{sub}(r^\bullet)$ implies $\varepsilon \notin \mathrm{L}(s)$. For expressions $\varnothing$, ε, and $a \in \mathcal{A}$, the claim is trivial, since the SSNF of such an expression is free of unary operators. Assume that the claim is true for any expression smaller than r. Recall that we showed monotonicity of $^\circ$ and $^\bullet$ in Prop. 3.2.1, thus if the inductive assumption applies to s, it also applies to s° and $s^\bullet$.

  For $r = st$, where $r^\bullet$ is either $s^\bullet t^\bullet$ or $s^\bullet + t^\bullet$, we have $\mathrm{sub}(r^\bullet) = \mathrm{sub}(s^\bullet) \cup \mathrm{sub}(t^\bullet)$. Since s and t, resp. $s^\bullet$ and $t^\bullet$, are smaller than r, the claim follows from the inductive assumption. For $r = s + t$, where $r^\bullet = s^\bullet + t^\bullet$, the same argument applies.

  For $r = s^?$, we examine two cases. If $\varepsilon \in \mathrm{L}(s)$, then $r^\bullet = s^\bullet$ by definition, and the claim follows since s is smaller than r. If $\varepsilon \notin \mathrm{L}(s)$, then $r^\bullet = s^{\bullet ?}$, thus $\mathrm{sub}(r^\bullet) = \mathrm{sub}(s^\bullet) \cup s^?$. For $\mathrm{sub}(s^\bullet)$ the inductive assumption applies again, whereas for $s^?$ the case assumes $\varepsilon \notin \mathrm{L}(s)$ already.

  Finally, for $r = s^*$ we have $r^\bullet = s^{\circ \bullet *}$, and thus $\mathrm{sub}(r^\bullet) = \mathrm{sub}(s^{\circ\bullet}) \cup s^{\circ\bullet *}$. Since s is smaller than r, so is $s^{\circ\bullet}$, and the claim follows for this subset of $\mathrm{sub}(r^\bullet)$. For the one remaining subexpression, we need to show $\varepsilon \notin \mathrm{L}(s^{\circ\bullet})$.

We know from Prop. 3.2.3 that $\varepsilon \notin \mathrm{L}(s^\circ)$, and since $s^{\circ\bullet} \equiv s^\circ$, by Lem. 3.2.4, this last bit follows, too.

Now if we assume $r = r^\bullet$ in the first place, we have that $\mathrm{sub}(r) = \mathrm{sub}(r^\bullet)$, and the claim follows for this direction.

- $1. \Leftarrow 2.$
This direction is proven by induction over the size of expressions wherein unary operands are not nullable. The expressions $\varnothing$, ε, and $a \in \mathcal{A}$ are of this kind, and each of them satisfies $r = r^\bullet$ by definition. Assume that the claim holds for all expressions smaller than r, which includes all proper subexpressions of r.

  For $r = st$ and $r = s + t$ we find, respectively,

$$r^\bullet = (st)^\bullet = s^\bullet t^\bullet \overset{I_A}{=} st = r \quad \text{and} \quad r^\bullet = (s+t)^\bullet = s^\bullet + t^\bullet \overset{I_A}{=} s + t = r.$$

  For $r = s^?$ the claim assumes $\varepsilon \notin \mathrm{L}(s)$, which implies $s \neq \varepsilon$, $s \neq t^?$, and $s \neq t^*$. Consequently, s is one of $\varnothing$, a letter, product, or sum, for each of which $s = s^\bullet$ was proven in already. So we find

$$r^\bullet = s^{?\bullet} = s^{\bullet ?} = s^? = r.$$

  For $r = s^*$ the structure of s is restricted to $\varnothing$, a letter, a product, or a sum, as in the previous case. Moreover, since $\varepsilon \notin \mathrm{L}(s)$ by assumption, Prop. 3.2.7 yields that $s^\circ = s$. This gets us

$$r^\bullet = s^{*\bullet} \overset{\text{def}}{=} s^{\circ\bullet *} = s^{\bullet *} \overset{I_A}{=} s^* = r.$$

□

Based on Thm. 3.2.8, we are able to define expressions in SSNF in a strictly syntactical fashion. For convenience, we do this in form of a Backus-Naur system again, where $\mathrm{RE}^\bullet$ denotes the class of regular expressions in SSNF, and where $\mathrm{RE}^\bullet_\varepsilon$ and $\mathrm{RE}^\bullet_{\not\varepsilon}$ denote the classes of nullable and non-nullable expressions in SSNF, respectively.

$$\begin{aligned}
\mathrm{RE}^\bullet_\varepsilon &::= \varepsilon \mid \mathrm{RE}^\bullet_\varepsilon\ \mathrm{RE}^\bullet_\varepsilon \mid \mathrm{RE}^\bullet_\varepsilon + \mathrm{RE}^\bullet_{\not\varepsilon} \mid \mathrm{RE}^\bullet_{\not\varepsilon} + \mathrm{RE}^\bullet_\varepsilon \mid \mathrm{RE}^{\bullet *}_{\not\varepsilon} \mid \mathrm{RE}^{\bullet ?}_{\not\varepsilon} \\
\mathrm{RE}^\bullet_{\not\varepsilon} &::= \varnothing \mid a \mid \mathrm{RE}^\bullet_{\not\varepsilon}\ \mathrm{RE}^\bullet_\varepsilon \mid \mathrm{RE}^\bullet_\varepsilon\ \mathrm{RE}^\bullet_{\not\varepsilon} \mid \mathrm{RE}^\bullet_{\not\varepsilon} + \mathrm{RE}^\bullet_{\not\varepsilon} \\
\mathrm{RE}^\bullet &::= \mathrm{RE}^\bullet_\varepsilon \mid \mathrm{RE}^\bullet_{\not\varepsilon}
\end{aligned}$$

The strong star normal form formalizes the intuition that the argument of a unary operator needs not be nullable, since * and $^?$ allow for ε by their "own" semantics. On the other hand, it is these two operators, together with instances of ε, that do allow for nullability of (sub)expressions, or operands. We consequently ask for the maximal number of unary operators an expression in SSNF might contain.

Theorem 3.2.9. *Let r be an expression in strong star normal form. Then*

$$|r|_\omega \leq \begin{cases} |r|_\mathcal{A} & \text{if } \varepsilon \in \mathrm{L}(r) \\ |r|_\mathcal{A} - 1 & \text{else} \end{cases}$$

Proof. The expressions $\varnothing$, ε, and $a \in \mathcal{A}$ are in SSNF and satisfy the claim. Let r be in SSNF and assume that the claim holds for every subexpression of r. Observe that with r each $s \in \mathrm{sub}(r)$ is in SSNF, too, so the inductive assumption applies to subexpressions of r.

If $r = st$ and r is nullable, both s and t are nullable, too. Thus

$$|st|_\omega = |s|_\omega + |t|_\omega \overset{I_A}{\leq} |s|_\mathcal{A} + |t|_\mathcal{A} = |st|_\mathcal{A},$$

as the statement claims for this case. On the other hand, if st is not nullable, then at least one out of s and t is not nullable. We assume wlog. that s is not nullable, to find

$$|st|_\omega = |s|_\omega + |t|_\omega \overset{I_A}{\leq} |s|_\mathcal{A} - 1 + |t|_\mathcal{A} = |st|_\mathcal{A} - 1.$$

For $r = s + t$, we distinguish once more whether r is nullable. If so, at least one out of s and t is nullable; again we assume wlog. that this is s. In this case we get

$$|s + t|_\omega = |s|_\omega + |t|_\omega \overset{I_A}{\leq} |s|_\mathcal{A} - 1 + |t|_\mathcal{A} = |s + t|_\mathcal{A} - 1.$$

Now if $s + t$ is not nullable, neither s nor t is, and the claim follows due to

$$|s + t|_\omega = |s|_\omega + |t|_\omega \overset{I_A}{\leq} |s|_\mathcal{A} + |t|_\mathcal{A} = |s + t|_\mathcal{A}.$$

If $r = s^*$ or $r = s^?$, then r is certainly nullable. Since r is in SSNF, Thm. 3.2.8 implies that s is not nullable in either case. Let $\mathrm{op} \in \{*, ?\}$, then the inductive step is

$$|s^{\mathrm{op}}|_\mathcal{A} = |s|_\mathcal{A} \overset{I_A}{\leq} |s|_\omega - 1 < |s^{\mathrm{op}}|_\omega,$$

and the proof is complete. □

Getting back to the simplification of regular expressions by reducing the number of unary operators, we combine Thm. 3.2.9 and the fundamental property of SSNF, to be a normal form. For any language we find an expression s.t. the size of this expression is bounded in the alphabetic width by a factor of three.

Theorem 3.2.10. *For any expression r there is an equivalent expression r' which satisfies $|r'| < 3|r'|_{\mathcal{A}}$.*

Proof. Let r be any expression, then $r^\bullet$ is equivalent to r, and, following Thm. 3.2.9, satisfies $|r^\bullet|_\omega \leq |r^\bullet|_{\mathcal{A}}$. We thus find

$$\begin{aligned} |r^\bullet| &= |r^\bullet|_{\mathcal{A}} + |r^\bullet|_\cdot + |r^\bullet|_+ + |r^\bullet|_\omega \\ &\leq |r^\bullet|_{\mathcal{A}} + |r^\bullet|_{\mathcal{A}} - 1 + |r^\bullet|_{\mathcal{A}} \\ &< 3|r^\bullet|_{\mathcal{A}}. \end{aligned}$$

□

This improves over the bound given by Ellul et al. [16], who bounded expression size within seven times the alphabetic width. As a further consequence of Thm. 3.2.9, we can also improve on the bound that we established for relative unary complexity in Prop. 3.1.1.

Theorem 3.2.11. *The relative unary complexity of any regular language L is bounded from above as follows:*

$$\omega(L) \leq 1.$$

Proof. Let L be regular and let r be an expression s.t. $\mathrm{L}(r) = L$ and $|r|_{\mathcal{A}} = \alpha(L)$. Then $\mathrm{L}(r^\bullet) = L$, too, and Thm. 3.2.9 yields that $|r^\bullet|_\omega \leq |r^\bullet|_{\mathcal{A}}$. Since $|r|_{\mathcal{A}} = |r^\bullet|_{\mathcal{A}}$, the assumption about r yields $|r^\bullet|_{\mathcal{A}} = \alpha(L)$; using the bound established in Thm. 3.2.9, we find

$$\omega(L) \leq \frac{|r^\bullet|_\omega}{\alpha(L)} \leq \frac{|r^\bullet|_{\mathcal{A}}}{|r^\bullet|_{\mathcal{A}}} = 1$$

□

This bound cannot be improved further for expressions over an unbounded alphabet. To see this, we consider expressions over a growing subset of integers. Let $\mathcal{A}_n := \{\widehat{k} \mid k \in \mathbb{N},\, k \leq n\}$, and set

$$\underline{n} := \prod_{0 \leq k \leq n} \widehat{k}^? \qquad \text{and} \qquad \overline{n} := \sum_{0 \leq k \leq n} \widehat{k}^*.$$

Lemma 3.2.12. $\omega(\mathrm{L}(\underline{n})) = 1$.

Proof. Any expression denoting $\mathrm{L}(\underline{n})$ contains each letter of $\mathcal{A}_n$ at least once. Hence, the alphabetic complexity of $\mathrm{L}(\underline{n})$ is n, as witnessed by $\underline{n}$. Let r be any expression attaining this width, i.e., assume that each letter of $\mathcal{A}_n$ occurs exactly once in r.

We observe that since $\mathrm{L}(\underline{n})$ is finite, r is free of iterations. Moreover, r is free of sums: suppose r contains a sum $s + t$, then the letters in s and t can not appear in the same word. This is seen as follows: since every letter of $\mathcal{A}_n$ appears exactly once in r, no letter appears both in s and in t. But since $s + t$ does not appear in the scope of an iteration, it is "evaluated" at most once, either in s or in t. Therefore, if r contains a sum, the word $\widehat{0}\widehat{1}\cdots\widehat{n}$, which contains every letter from $\mathcal{A}_n$, is not contained in $\mathrm{L}(r)$.

So the binary operators in r are all concatenations. Now suppose that r contains less than n option symbols. Then either there is a letter $\widehat{i}$ in r that is not in the scope of an option at all, or there are two letters $\widehat{j}$ and $\widehat{k}$ that are in the scope of an option together. In the first case, $\widehat{i}$ is present in each $w \in \mathrm{L}(r)$, whereas in the second, $\widehat{j}$ and $\widehat{k}$ are either both present or both absent in each $w \in \mathrm{L}(r)$. Since this does not reflect $\mathrm{L}(\underline{n})$ adequately, r must contain at least n option symbols.

For the relative unary complexity of $\mathrm{L}(\underline{n})$ we thus find $\omega(\mathrm{L}(n)) \geq 1$, which, together with Thm. 3.2.11, yields the statement. □

The statement is proven similarly for $\mathrm{L}(\overline{n})$, essentially by showing that each letter in an expression that reaches the alphabetic complexity of the language is the exclusive argument of at least one iteration. Later, in Ch. 5, we will consider an infinite family of expressions which demonstrates the lower bound on unary complexity for languages over a finite alphabet.

4 Converting Expressions to Automata

In this chapter we investigate the conversion of regular expressions into finite automata, which is arguably the more important one among the translation between expressions and automata. This is due to the fact that regular languages — search patterns, in particular — are usually specified as regular expressions, which come natural to humans. On the machine level, however, the data structure that efficiently represents such a language, is a finite automaton. A plenitude of constructions of finite automata from regular expressions have been proposed over the last half century, starting with Kleene's 1956 work wherein the expressive equivalence of expressions and automata is shown constructively [31], i.e., by giving mutual conversions.

The various methods to construct automata from regular expression can be coarsely separated into the "algebraic" and the "recursive" approach. The first type of construction is based on the Myhill-Nerode equivalence and yields deterministic automata. As we remarked in Ch. 3, these automata might be of exponential size relative to the size of the input expression, thus the running time of these constructions is bounded exponentially from below. The second type of construction produces an automaton by traversing the parse of the input expression in a top-down or bottom-up manner either combining subautomata or introducing new automaton components along the process. Automata that are constructed recursively from expressions are usually not deterministic. These constructions can be subcategorized further by whether the resulting automata contain ε-transitions. If ε-transitions are allowed, the construction is usually linear in expression size, whereas if not, the construction may require quadratic time. Various constructions of either type are compiled and categorized in Watson's excellent survey [55].

The construction given in this chapter is of the recursive type. It works by processing an expression in a top-down fashion, introducing states and transitions to an intermediate EFA with each advance down the parse of the input expression. These steps constitute the core of the construction; they are borrowed from an earlier conversion algorithm that was proposed by Ott & Feinstein [41]. The presented construction applies further steps that hold the

number of ε-transitions down to a minimum, as far as can be inferred from the input without sacrificing running time. The results presented in this and the following chapter are based on [22, 21]. However the present work heavily extends — and sometimes corrects — these former results.

4.1 Expansions and Eliminations

We introduce a set of relations on EFAs, which, after some technical modifications, will become the rules of a rewriting system. These relations are roughly divided into *expansions* and *eliminations*. Expansions are defined relative to the root of transition labels in an EFA, and they are independent of graph structure of the considered EFA. Eliminations are defined for certain substructures that contain ε-transitions; in contrast to expansions, they do depend on the graph structure.

Definition 3 (Expansion)**.** Let $E = (Q, \mathcal{A}, \delta, I, F)$ be an EFA. The relations $\Rightarrow_\varnothing$, $\Rightarrow_.$, $\Rightarrow_+$ and $\Rightarrow_*$, called *zero expansion*, *product expansion*, *sum expansion*, and *star expansion*, respectively, are defined on **EFA** as follows:

- Zero expansion:
  Let E contain a transition $t = (p, \varnothing, q) \in \delta$. Then $E \Rightarrow_\varnothing E'$ holds for

  $$E' = (Q, \mathcal{A}, \delta \setminus t, I, F).$$

  We say that E is *zero expanded* to E' in t.

- Product expansion:
  Let E contain a transition $t = (p, st, q) \in \delta$. Then $E \Rightarrow_. E'$ holds for

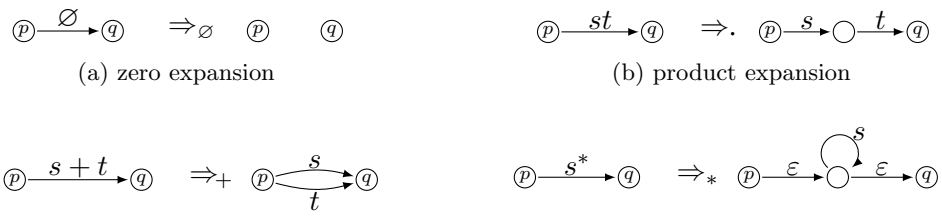
  $$E' = (Q \uplus p', \mathcal{A}, \delta \setminus t \cup \{(p, r, p'), (p', s, q)\}, I, F).$$

  We say that E is *product expanded* to E' in t.

- Sum expansion:
  Let E contain a transition $t = (p, s + t, q) \in \delta$. Then $E \Rightarrow_+ E'$ holds for

  $$E' = (Q, \mathcal{A}, \delta \setminus t \cup \{(p, s, q), (p, t, q)\}, I, F).$$

  We say that E is *sum expanded* to E' in t.



(a) zero expansion

(b) product expansion

(c) sum expansion

(d) star expansion

Figure 4.1: Expanding a transition (p, r, q) depending on r.

- Star expansion:
  Let E contain a transition $t = (p, s^*, q) \in \delta$. Then $E \Rightarrow_* E'$ holds for

$$E' = (Q \cup p', \mathcal{A}, \delta \setminus t \cup \{(p, \varepsilon, p'), (p', s, p'), (p', \varepsilon, q)\}, I, F).$$

  We say that E is *star expanded* to E' in t.

The local changes in E are sketched in Fig. 4.1 for each type of expansion. The transition which allows for an expansion, is also referred to as the *anchor* of this expansion. Notice that we did not include a rule to handle transitions that are labeled with an option. The expression $r^?$ shall be implicitly treated as $r + \varepsilon$, i.e., "option expansion" is considered to be a special case of sum expansion. We set

$$\Rightarrow_{\text{ex}} := \Rightarrow_\varnothing \cup \Rightarrow_. \cup \Rightarrow_+ \cup \Rightarrow_*$$

and write just $E \Rightarrow_{\text{ex}} E'$ if we do not care for the details of the expansion from E to E'. We do, however, introduce an alternative notation for expansions. Notice that the effects of any expansion are restricted to a local part of an EFA. In particular, an expansion is fully specified by its anchor: the states incident to this transition determine the location of the replacement, and its label determines the elements that are introduced or removed. We convey this information by writing $E \Rightarrow_{[t]} E'$ if E is expanded to E' in t.

Definition 4 (Elimination)**.** Let $E = (Q, \mathcal{A}, \delta, I, F)$ be an EFA. The relations $\Rightarrow_\circlearrowleft$, $\Rightarrow_\triangleleft$, $\Rightarrow_X$, and $\Rightarrow_Y$ are called *cycle elimination*, *fan elimination*, *X elimination*, and *Y elimination*, respectively. They are defined as follows:

- cycle elimination:
  Let E contain an ε-cycle γ, with states $Q_\gamma = \{q_1, \ldots, q_n\}$ and transitions $\delta_\gamma = \{(q_i, \varepsilon, q_{i+1}) \mid 1 \leq i < n\} \cup (q_n, \varepsilon, q_1)$. Then $E \Rightarrow_\circlearrowleft E'$ holds for

$$E' = ((Q \setminus Q_\gamma) \uplus q_\gamma, \mathcal{A}, \delta', I', F'),$$

where

$$\begin{aligned}\delta' = (\delta \setminus \delta_\gamma) \cup \{(p, r, q_\gamma) \mid (p, r, q) \in \delta \setminus \delta_\gamma,\, q \in Q_\gamma\} \\ \cup \{(q_\gamma, r, p) \mid (q, r, p) \in \delta \setminus \delta_\gamma,\, q \in Q_\gamma\}\end{aligned}$$

$$I' = \begin{cases}(I \setminus Q_\gamma) \cup q_\gamma, & \text{if } Q_\gamma \cap I \neq \emptyset \\ I, & \text{otherwise}\end{cases}$$

$$F' = \begin{cases}(F \setminus Q_\gamma) \cup q_\gamma, & \text{if } Q_\gamma \cap F \neq \emptyset \\ F, & \text{otherwise}\end{cases}$$

- fan elimination:
  Let E contain a state q with $\mathrm{d}^-(q) = 1$ and $(p, \varepsilon, q) \in \delta$ for some $p \neq q$. Then the relation $E \Rightarrow_\lhd E'$ holds for

$$E' = (Q \setminus q, \mathcal{A}, \delta', I', F'),$$

  where

$$\delta' = \delta \setminus ((p, \varepsilon, q) \cup \{(q, r, q') \mid (q, r, q') \in \delta\}) \cup \{(p, r, q') \mid (q, r, q') \in \delta\},$$

$$I' = \begin{cases}(I \setminus q) \cup p, & \text{if } q \in I \\ I, & \text{otherwise}\end{cases}$$

$$F' = \begin{cases}(F \setminus q) \cup p, & \text{if } q \in F \\ F, & \text{otherwise}\end{cases}$$

- X-elimination:
  Let E contain a state q with $\mathrm{d}^-(q) = \mathrm{d}^+(q) = 2$, incident to states p_1, p_2, p_3, and p_4 via transitions $t_1 = (p_1, \varepsilon, q)$, $t_2 = (p_2, \varepsilon, q)$, $t_3 = (q, \varepsilon, p_3)$, and $t_4 = (q, \varepsilon, p_4)$. Then the relation $E \Rightarrow_X E'$ holds for

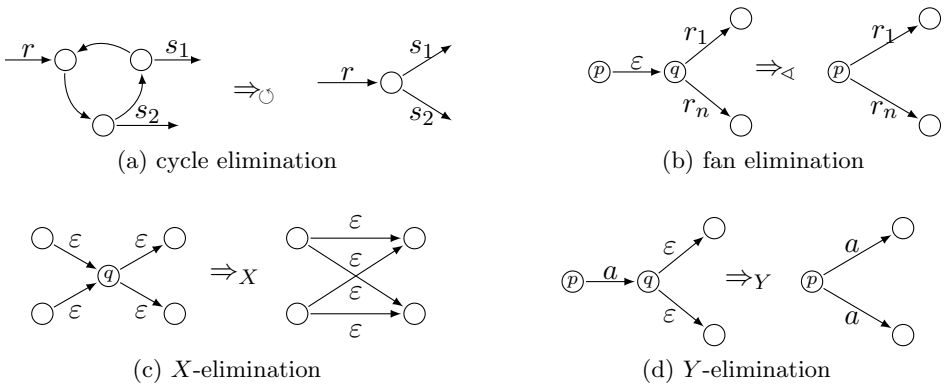
$$E' = (Q \setminus q, \mathcal{A}, \delta, I', F'),$$

  where

$$\delta' = \delta \setminus \{t_1, t_2, t_3, t_4\} \cup \{(p_1, \varepsilon, p_3), (p_1, \varepsilon, p_4), (p_2, \varepsilon, p_3), (p_2, \varepsilon, p_4)\}$$

$$I' = \begin{cases}I \setminus q \cup \{p_3, p_4\}, & \text{if } q \in I \\ I, & \text{otherwise}\end{cases}$$

$$F' = \begin{cases}F \setminus q \cup \{p_1, p_2\}, & \text{if } q \in F \\ F, & \text{otherwise}\end{cases}$$



(a) cycle elimination (b) fan elimination

(c) X-elimination (d) Y-elimination

Figure 4.2: Elimination of substructures with ε-transitions.

- Y-elimination:
  Let $q \in Q_E$ s.t. $\mathrm{d}^-(q) = 1$ for a single transition $t = (p, a, q)$ with $a \in \mathcal{A}$. Suppose further that all transitions leaving q are ε-transitions. The relation $E \Rightarrow_Y E'$ holds for

$$E' = (Q \setminus q, \mathcal{A}, \delta, I', F'),$$

  where

$$\delta' = \delta \setminus (t \cup \delta_q) \cup \{(p, a, q_i) \mid (q, \varepsilon, q_i) \in \delta_E\},$$

$$I' = \begin{cases} I \setminus q \cup p, & \text{if } q \in I \\ I, & \text{otherwise} \end{cases}$$

$$F' = \begin{cases} F \setminus q \cup p, & \text{if } q \in F \\ F, & \text{otherwise} \end{cases}$$

The local changes in an EFA upon elimination are sketched in Fig. 4.2. As for expansions, we refer to the part of an EFA that is replaced by a particular elimination, i.e., any left hand side in Fig. 4.2, as the *anchor* of this elimination.

Observe that for fan elimination, X-elimination, and Y-elimination, an anchor consists of a single state and all its incident transitions. We call this state the *center* of the anchor[1]. When convenient, the center q of an elimination will be

[1] In fact, these three rewritings are particular cases of *state elimination*, a method originally proposed by Brzozowski & McCluskey for the dual construction: the conversion of automata to expressions [10].

conveyed in formal notation by writing $\Rightarrow_{\lhd[q]}$, $\Rightarrow_{X[q]}$, or $\Rightarrow_{Y[q]}$. In analogy to $\Rightarrow_{\mathrm{ex}}$ for expansions, we set

$$\Rightarrow_{\mathrm{el}} := \Rightarrow_{\circlearrowleft} \cup \Rightarrow_{\lhd} \cup \Rightarrow_X \cup \Rightarrow_Y,$$

and write $E \Rightarrow_{\mathrm{el}} E'$, to denote an arbitrary elimination from E to E'. So far, we have arrived at the ARS

$$\mathfrak{C} := \langle \mathbf{EFA}, \Rightarrow_\varnothing, \Rightarrow_., \Rightarrow_+, \Rightarrow_*, \Rightarrow_{\circlearrowleft}, \Rightarrow_{\lhd}, \Rightarrow_X, \Rightarrow_Y \rangle,$$

or just $\mathfrak{C} = \langle \mathbf{EFA}, \Rightarrow_{\mathrm{ex}}, \Rightarrow_{\mathrm{el}} \rangle$. It should be clear from Figs. 4.1 and 4.2 that the language accepted by an EFA is invariant under rewritings in $\mathfrak{C}$. We thus omit proving

Proposition 4.1.1. *If $E \Rightarrow_{\mathrm{ex}} E'$ or $E \Rightarrow_{\mathrm{el}} E'$, then $\mathrm{L}(E) = \mathrm{L}(E')$.*

We employ $\mathfrak{C}$ for the conversion of expressions into automata. To this end, we first identify an expression with a structurally trivial EFA that accepts the language denoted by that expression.

Definition 5. Let $r \in \mathrm{RE}_{\mathcal{A}}$. The *primal* EFA associated to r is

$$A_r^0 := (\{q_0, q_f\}, \mathcal{A}, (q_0, r, q_f), q_0, q_f).$$

Further let A_r^n denote any EFA that can be derived from A_r^0 by a rewriting of length n in $\mathfrak{C}$. Each step of a rewriting initiated in A_r^0 is thus of the form

$$A_r^n \Rightarrow_{\mathfrak{C}} A_r^{n+1}$$

for some n. Any $\mathfrak{C}$-normal form of A_r^0 will be denoted simply A_r. Notice that A_r^n is generally not unique for $n \geq 1$, neither is A_r. Since A_r is normal wrt. every expansion, all transitions of A_r are labeled by ε or a letter. Therefore $\mathfrak{C}$ realizes the conversion of expressions into automata:

Proposition 4.1.2. *$r \equiv A_r$ for every $r \in \mathrm{RE}$.*

Proof. The equivalence $r \equiv A_r^0$ holds trivially. Now Prop. 4.1.1 yields $r \equiv A_r^n$, in particular $r \equiv A_r$. □

Every transition of each EFA derived from A_r^0 in $\mathfrak{C}$ is labeled with a subexpression of r. Conversely, in a full rewriting $A_r^0 \Rightarrow^\star A_r$, every subexpression of r will at some point become a label of a transition. Moreover, replacing subexpressions of r leads to corresponding replacements of labels in A_r^n. We formalize this by first assuming that a subexpression is unique.

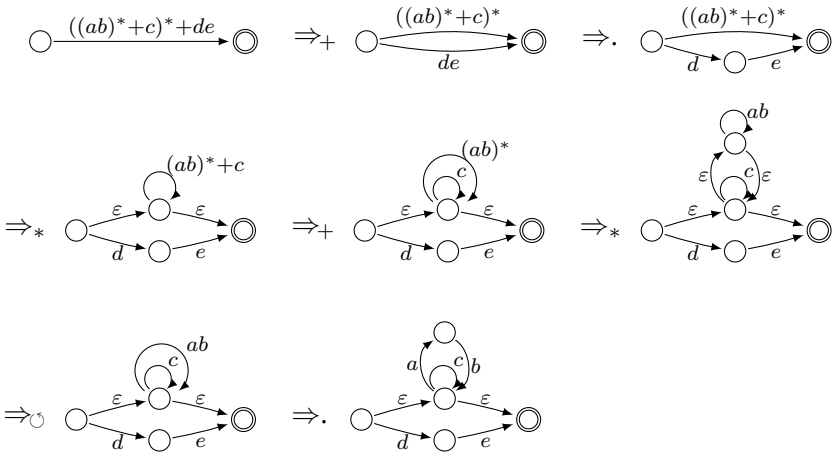


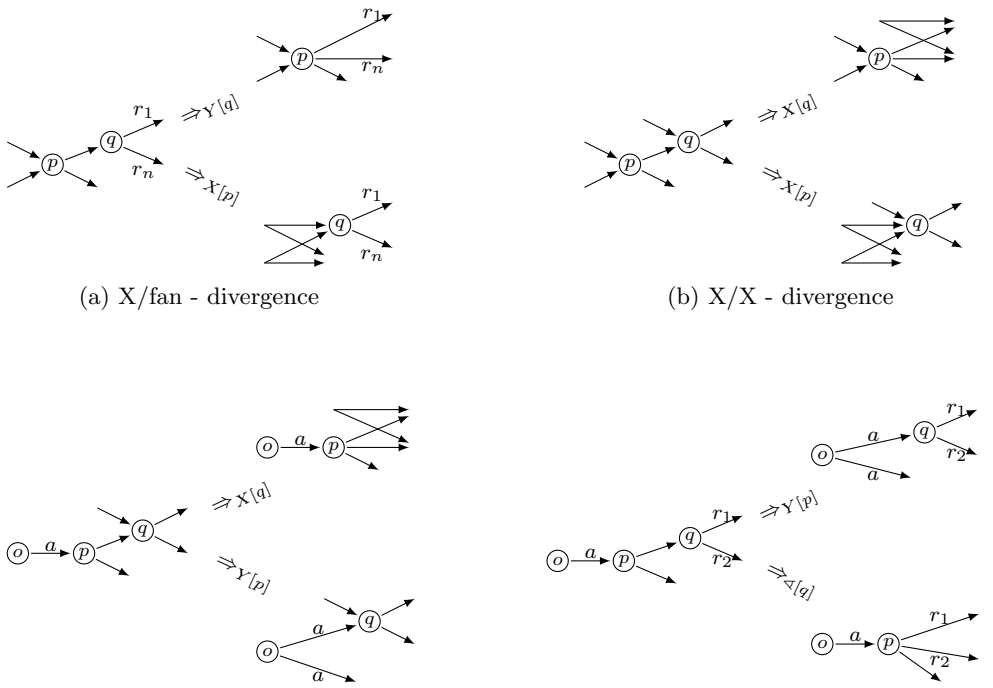
Figure 4.3: Example of a full conversion from A_r^0 to A_r.

Proposition 4.1.3. *Let $r = c[s]$ with just one occurrence of s and suppose $t = (p, s, q)$ is a transition of A_r^n. For $r' = c[s']$ there is a rewriting $A_{r'}^0 \Rightarrow^\star A_{r'}^n$ s.t. $A_{r'}^n$ is as A_r^n except that t is replaced with $t' = (p, s', q)$.*

Proof. Straightforward induction over the length of the rewriting from A_r^0 to A_r^n, which is n. □

In the case that is formulated in Prop. 4.1.3, the subexpression that provides the considered label is uniquely determined. More generally, we could label transitions with the position of a subexpression, instead of the subexpression itself. Recall that positions allow to distinguish different occurrences of a subexpression; this carries over to transitions with the same label in some A_r^n. With positions as labels we could rephrase Prop. 4.1.3 to apply to arbitrary subexpressions. However, the notational overhead in doing so is considerable and this approach will not be pursued formally.

Still, we implicitly use the more general variant of Prop. 4.1.3 in this chapter and Ch. 5 to some extent, by comparing automata that are constructed from expressions that differ in a replaced subexpression that is not necessarily unique. It is possible to enforce the preconditions of Prop. 4.1.3, e.g. by tagging a letter in the subexpression of interest, thereby switching to a different expression (over a bigger alphabet) with the same structure.



(a) X/fan - divergence

(b) X/X - divergence

(c) X/Y - divergence

(d) Y/fan - divergence

Figure 4.4: Divergences from X- and Y-eliminations. Unlabeled arrows depict ε-transitions, and a is a letter.

4.2 Refining the ARS to Functionality

As noticed, the FA constructed from r by rewriting A_r^0 in $\mathfrak{C}$ is generally not unique, i.e., A_r might depend on the particular rewriting sequence. Thus the ARS $\mathfrak{C}$ is not functional in the sense of realizing a map from expressions to automata. While this is irrelevant if one is merely interested in a linear-time construction of automata from expressions, it hinders further analysis of the obtained automata.

It is obvious that $\mathfrak{C}$ is terminating. Hence, following Newman's Lemma, the reason why A_r is not unique is that $\mathfrak{C}$ is not locally confluent. As it turns out, every divergence that does not converge in $\mathfrak{C}$, involves an X- or an Y-elimination. Characteristic examples leading to non-convergent divergences are shown in Fig. 4.4.

In the following, we separate X- and Y-elimination from the other rewritings in

order to guarantee unique normal forms. For one thing, the rewritings besides $\Rightarrow_X$ and $\Rightarrow_Y$ are treated in a common ARS. This sub-ARS of $\mathfrak{C}$ is called the *basic* rewriting system, it is defined as

$$\mathfrak{B} := \langle \mathbf{EFA}, \Rightarrow_\varnothing, \Rightarrow_., \Rightarrow_+, \Rightarrow_*, \Rightarrow_\circlearrowleft, \Rightarrow_\triangleleft \rangle.$$

For each of X- and Y-elimination we consider a separate sub-ARS of $\mathfrak{C}$. These systems will be defined over a restricted universe and use restricted "variations" of $\Rightarrow_X$ and $\Rightarrow_Y$ as rules. To define these variations, we say that an $X[p]$-anchor *dominates* an $X[q]$-anchor in E, if E contains a transition (p, ε, q). Likewise, a $Y[p]$-anchor *dominates* a $Y[q]$-anchor if there is an ε-transition from p to q.

Definition 6. An X-elimination $E \Rightarrow_{X[q]} E'$ is *valid*, if the following conditions are met:

(X1) E is $\mathfrak{B}$-normal.

(X2) The X[q]-anchor in E is not dominated by some other X-anchor.

Likewise, a Y-elimination $E \Rightarrow_{Y[q]} E'$ is *valid*, if the following conditions are met:

(Y1) E is $\mathfrak{B}$-normal and X-normal.

(Y2) The Y[q]-anchor in E is not dominated by some other Y-anchor.

Observe that **(X2)** restricts the relation $\Rightarrow_X$ based on the simultaneous presence of various X-anchors. Still, every sequence of (valid) X-eliminations eventually terminates in an X-normal FA. This is because a cyclic domination among X-anchors, i.e., an "elimination deadlock", implies the existence of an ε-cycle, which, due to **(X1)**, is eliminated before valid X-elimination is applicable. The respective property holds for (valid) Y-eliminations.

Lemma 4.2.1.

1. *If $E \Rightarrow_X E'$ is valid, then E' is $\mathfrak{B}$-normal.*
2. *If $E \Rightarrow_Y E'$ is valid, then E' is $\mathfrak{B}$-normal and X-normal.*

Proof. We prove the first claim, the second is shown analogous. If $E \Rightarrow_{X[q]} E'$ is valid, then E is $\mathfrak{B}$-normal, so E contains no expansion-anchors, ε-cycles or fans. If any such anchor appears in E', it must be a result of the preceding X-elimination. However, neither $\varnothing$ nor any compound label, i.e., no expansion

anchor, can be introduced by X-elimination alone. Moreover, if E' contains an ε-cycle with n transitions, E necessarily contains an ε-cycle on $n+1$ transitions, contradicting the assumption that E is $\mathfrak{B}$-normal. Suppose finally that E' contains a fan centered in some state q' with $\mathrm{d}^-_{E'}(q') = 1$ and an ε-transition (p, ε, q') (the case $\mathrm{d}^+_{E'}(q') = 1$ is symmetric). As this fan emerged from local changes in E upon elimination, q' must be incident to q in E. Now, X-elimination increases either the in- or the out-degree of a state incident to the center ot this elimination, thus the incidence of q' and q in E is due to an ε-transition leaving q. This yields us $\mathrm{d}^-_E(q') = 1$, too. Consequently, E contains a fan in q' and is hence not $\mathfrak{B}$-normal. This contradicts the initial assumption that $E \Rightarrow_X E'$ is valid.

We have shown that only X- and Y-anchors are present in E', if any. It follows that E' is $\mathfrak{B}$-normal, as claimed. □

Corollary 4.2.2. *Let $E \Rightarrow^\star F$ be a valid rewriting in $\mathfrak{C}$. Then there are EFAs E' and E'' s.t.*

$$E \Rightarrow^\star_{\mathfrak{B}} E' \Rightarrow^\star_X E'' \Rightarrow^\star_Y F.$$

Proof. The first X-elimination of $E \Rightarrow^\star F$ is applied to a normal form of E in $\mathfrak{B}$, as demanded by **(X1)**. Following Lem. 4.2.1, applying X-elimination to a $\mathfrak{B}$-normal EFA yields a further $\mathfrak{B}$-normal EFA. Likewise, since $E \Rightarrow^\star F$ also respects **(Y1)**, the first Y-elimination is applied to an EFA that is both $\mathfrak{B}$-normal and X-normal. Again, Lem. 4.2.1 shows that no other type of rewriting can possibly follow the first Y-elimination. □

Thus, restricting X- and Y-eliminations to valid instances separates the rewritings in the basic ARS $\mathfrak{B}$ from those eliminations. However, Lem. 4.2.1 does *not* imply that E' is $\mathfrak{B}$-normal or that E'' is $\mathfrak{B}$- and X-normal; notice that the proof refers to the *first* application of X- resp. Y-elimination.

Let $E \Rrightarrow_X E'$ denote the valid X-elimination $E \Rightarrow_X E'$. It follows from Lem. 4.2.1 that $\Rrightarrow_X$ is a relation on the class of FAs that are free of fans and ε-cycles. Let $\mathbf{FA_X}$ denote this class. We formalize valid X-eliminations in the ARS

$$\mathfrak{X} := \langle \mathbf{FA_X}, \Rrightarrow_X \rangle.$$

Likewise, Y-elimination is a relation on the class we denote $\mathbf{FA_Y}$, which contains exactly the FAs without fans, ε-cycles, and X-anchors. The valid Y-eliminations

give rise to the rewriting system

$$\mathfrak{Y} := \langle \mathbf{FA_Y}, \Rrightarrow_Y \rangle.$$

Notice that $\Rrightarrow_X$ and $\Rrightarrow_Y$ are still just relations on **EFA**. Instead of replacing $\Rightarrow_X$ with $\Rrightarrow_X$ and $\Rightarrow_Y$ with $\Rrightarrow_Y$ in $\mathfrak{C}$, we proceed with the "modular" approach by splitting this potential ARS into $\mathfrak{B}$, $\mathfrak{X}$, and $\mathfrak{Y}$. This provides us with a better overview of the rewriting process as its independent parts — the rewritings due to $\mathfrak{B}$, $\mathfrak{X}$, and $\mathfrak{Y}$— are treated independently.

In the following, we prove that each of $\mathfrak{B}$, $\mathfrak{X}$, and $\mathfrak{Y}$ is locally confluent. This requires to investigate elements that are part of several anchors and can thus be modified in different ways. Formally, the *overlap* of two anchors consists of the elements shared by these anchors, i.e., the states and transitions that belong to both anchors. An overlap is *trivial*, if the involved anchors are identical or do not share any elements at all. Trivial overlaps are irrelevant in the analysis of confluence properties, as the following proposition states for the ARS $\mathfrak{B}$.

Proposition 4.2.3. *Let* $E \Rightarrow_{\mathrm{i}} E_1$ *and* $E \Rightarrow_{\mathrm{j}} E_2$ *for* $\mathrm{i}, \mathrm{j} \in \{\varnothing, \cdot, +, *, \circlearrowleft, \lhd\}$. *If the overlap of the corresponding* i- *and* j*-anchor is trivial, then* $E_1 \sim_{\mathfrak{B}} E_2$.

Proof. If the two anchors are identical, then $E_1 = E_2$ and the statements follows. If the anchors share neither states nor transitions, the rewritings happen in different "regions" of E and do not interfere with each other. Thus E_1 contains the same j-anchor as E while E_2 contains the same j-anchor as E. So for some E_3 we find

$$E_1 \Rightarrow_{\mathrm{j}} E_3 \quad \text{and} \quad E_2 \Rightarrow_{\mathrm{i}} E_3.$$

□

The convergence of EFAs derived from non-overlapping rewritings thus holds in each of $\mathfrak{B}$, $\mathfrak{X}$, and $\mathfrak{Y}$. Henceforth we thus assume that overlaps are nontrivial. Other than that, two anchors of a fixed type might allow for structurally different kinds of overlaps that need to be treated on a case by case basis. This will become relevant for eliminations; in the case that two expansions are considered, the argument is fairly simple.

Lemma 4.2.4. *For* $E \Rightarrow_{\mathrm{ex}} E_1$ *and* $E \Rightarrow_{\mathrm{ex}} E_2$ *we find* $E_1 \sim_{\mathfrak{B}} E_2$.

Proof. The effects of expanding E in a given transition are independent of any property of E. Moreover, expansion does not alter any state of E nor any transition besides that which is expanded. Thus the order of two successive expansions is irrelevant. Formally, t is a transition of E_2 and t' is a transition of E_1, and there is a unique EFA E_3 s.t.

$$E \Rightarrow_{[t]} E_1 \Rightarrow_{[t']} E_3 \text{ and } E \Rightarrow_{[t']} E_2 \Rightarrow_{[t]} E_3$$

are rewritings in $\mathfrak{B}$. □

Therefore, exhaustive expansion of A_r^0 yields a unique FA. This method to construct FAs from expressions was proposed as early as 1961 by Ott & Feinstein [41]. It provides FAs that are considerably smaller than those yielded by Thompson's algorithm [51], which introduces an abundance of ε-transitions. Although Ott & Feinstein's algorithm precedes Thompson's by seven years, the latter is usually cited as the canonical construction of FAs based on expression structure [49, 55, 43].

We extend Lem. 4.2.4 to rewritings with cycle and fan eliminations, covering all rules of $\mathfrak{B}$. The proofs are considerably more involved — although not harder — than for Lem. 4.2.4. Because of this we establish the local confluence of $\mathfrak{B}$ piecewise.

Lemma 4.2.5. *If $E \Rightarrow_{\circlearrowleft} E_1$ and $E \Rightarrow_{\mathfrak{B}} E_2$, then $E_1 \sim_{\mathfrak{B}} E_2$.*

Proof. Let γ denote the ε-cycle which is eliminated upon $E \Rightarrow_{\circlearrowleft} E_1$ and let Q_γ and T_γ denote the states and transitions of γ. In this proof we take a different view on cycle elimination compared to its definition; it is easily seen that the two notions are equivalent. Cycle elimination can be described as follows: choose a state $q_r \in Q_\gamma$, called the *representative*, and redirect all transitions in $\delta_E \setminus T_\gamma$ from or to states in $Q_\gamma \setminus q_r$ to leave or enter q_r, respectively. Then remove T_γ and $Q_\gamma \setminus q_r$, adjusting I_E and F_E accordingly. This yields E_1.

The main distinction of this proof is by the type of the rewriting $E \Rightarrow_{\mathfrak{B}} E_2$.

- If E_2 is obtained from E by expansion, let $E \Rightarrow_{[t]} E_2$ for $t = (p, r, q)$. A nontrivial overlap of $\Rightarrow_{\circlearrowleft}$ and $\Rightarrow_{[t]}$ may consist only of p or q (or both), since the label of t is a compound label by assumption. We assume $p \in Q_\gamma$ wlog. and choose p as the representative of γ for cycle elimination. Then E_1 contains the transition $t' = (p, r, q')$, where $q' = q$, and thus $t' = t$, if $q \notin Q_\gamma$, or $q' = p$ otherwise. In either case, E_1 can be expanded via $E_1 \Rightarrow_{[t']} E_3$, where $\{p, q'\}$ is still a (possibly singleton) set of states of E_3.

On the other hand, $E \Rightarrow_{[t]} E_2$ does not alter the ε-cycle γ in any way whatsoever. Hence γ is an ε-cycle in E_2, and moreover, p and q are states of E_2, with $p \in Q_\gamma$. Again we may choose p as the representative for cycle elimination $E_2 \Rightarrow_\gamma E_3'$. Then E_3' contains p, as well as q' which is distinct from p iff q was no part of γ in E, resp. E_2.

- If E_2 results from eliminating a further cycle γ' in E, assume that the two cycles overlap and let q be a state of either cycle. We argue that there is an EFA E_3, reached from E_1 as well as E_2 by a sequence of cycle eliminations. E_3 can be constructed from E removing the states except q and transitions of γ and γ', and redirecting transitions connected to the cycles appropriately. If the overlap of γ with γ' consists of q alone, this follows easily, since γ' is present in E_1 while γ is present in E_2. We then find

$$E \Rightarrow_\gamma E_1 \Rightarrow_{\gamma'} E_3 \quad \text{and} \quad E \Rightarrow_{\gamma'} E_2 \Rightarrow_\gamma E_3.$$

  Next, consider the case that γ and γ' share more than one state. As γ and γ' are distinct by assumption, there are transitions $t_1', \ldots, t_n' \in T_{\gamma'} \setminus T_\gamma$ which form a path from p to q for arbitrary $p, q \in Q_\gamma \cap Q_{\gamma'}$. These transitions are also present in E_1, which is obtained from E by eliminating γ, except that the head of t_1' and the tail of t_n' are relabeled q, the representative of γ. In other words, the t_i' form an ε-cycle in E_1. A cycle arises for every sequence as described, and we may choose q as the representative state for every such cycle. If there are k sequences of this type, forming cycles $\gamma_1', \ldots, \gamma_k'$ in E_1, let E_3 be the EFA satisfying

$$E \Rightarrow_\gamma E_1 \Rightarrow_{\gamma_1'} \cdots \Rightarrow_{\gamma_k'} E_3.$$

  Thus all states of $Q_\gamma \cup Q_{\gamma'}$ are merged into q in E_3 while the transitions $T_\gamma \cup T_{\gamma'}$ are removed. The same happens if γ' is removed first and the remaining transitions of γ are sequentially eliminated in E_2, i.e.,

$$E \Rightarrow_{\gamma'} E_2 \Rightarrow_{\gamma_1} \cdots \Rightarrow_{\gamma_l} E_3.$$

- If E_2 results from eliminating a fan with center q, assume that this fan is given by the in-transition $t = (p, \varepsilon, q)$ and out-transitions $t_i = (q, r_i, q_i)$ for $1 \leq i \leq n$. We distinguish whether the center of the fan lies on the ε-cycle.

  If $q \notin Q_\gamma$, then neither t nor any t_i is contained in T_γ. We choose p as the representative of γ for cycle elimination, then $t = (p, \varepsilon, q)$ is also a transition of E_1. Also, for every t_i in E, E_1 contains transition $t_i' = (q, r_i, q_i')$, where $q_i' = p$ if $q_i \in Q_\gamma$ in E and $q_i' = q_i$ otherwise. The in-degree of q certainly

does not increase with cycle elimination, so q is the center of a fan in E_1, too. Eliminating this fan yields E_3, which can also be obtained by first applying fan elimination, which leaves γ intact, followed by cycle elimination. In other words, the order of eliminations can be swapped without altering the resulting EFA. This case is sketched in Fig. 4.5a.

For $q \in Q_\gamma$, we show that cycle elimination subsumes fan elimination, whereas conversely, fan elimination can be considered as a "step" in the elimination of γ. As for further overlapping elements of the two anchors, notice that the condition $\mathrm{d}^-(q) = 1$ implies $t \in T_\gamma$. Moreover, the transition leaving q in γ is also part of the fan in consideration. Notice that fan elimination requires $p \neq q$, so γ contains at least two transitions. The EFA E_2, yielded by $E \Rightarrow_{\triangleleft q} E_2$, thus contains an ε-cycle γ' where $Q_{\gamma'} = Q_\gamma \setminus q$ and $T_{\gamma'} = T_\gamma \setminus \{t, t'\} \cup (p, \varepsilon, q')$, where q' is the state following q in γ, and $t' = (q, \varepsilon, q')$ is the corresponding transition. For γ' we choose p as the representative, so elimination of γ merges all states of $Q_{\gamma'}$ into p, which already happened for q in the preceding fan elimination. So in this case we have

$$E \Rightarrow_{\circlearrowleft} E_1, \text{ and } E \Rightarrow_{\triangleleft} E_2 \Rightarrow_{\circlearrowleft} E_1.$$

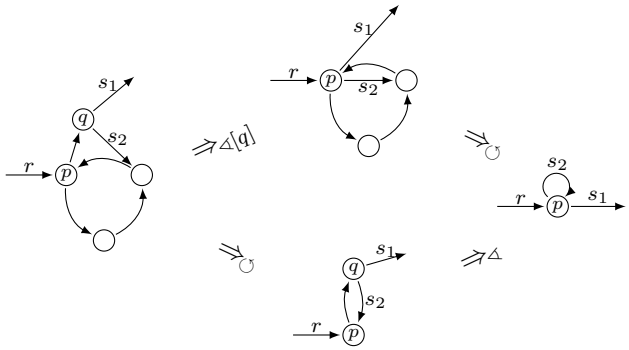
This case is shown in Fig. 4.5b.

All possibilities for an ε-cycle overlapping with some further anchor have been considered; in each case, we have found that the divergent EFAs rewrite to a common follow-up EFA. Thus the claim follows. □

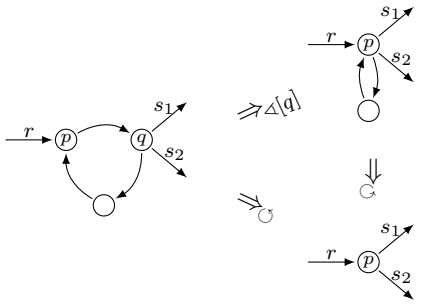
Lemma 4.2.6. *If $E \Rightarrow_{\triangleleft} E_1$ and $E \Rightarrow_{\mathfrak{B}} E_2$, then $E_1 \sim_{\mathfrak{B}} E_2$.*

Proof. Suppose the fan elimination is $E \Rightarrow_{\triangleleft[p]} E_1$, for an in-fan with center p, which is incident to transitions $t_\varepsilon = (p', \varepsilon, p)$ and $t_i = (p, r_i, q_i)$ for $1 \leq i \leq n$ (the argument is symmetric for an out-fan). As for the rewriting $E \Rightarrow_{\mathfrak{B}} E_2$, the case $E \Rightarrow_{\circlearrowleft} E_2$ was dealt with in Lem. 4.2.5 already. It remains to consider $E \Rightarrow_{\mathrm{i}} E_2$ for $\mathrm{i} \in \{\varnothing, \cdot, +, *, \triangleleft\}$.

Whether p is the center of a fan is determined by the property that p has only one incoming transition, which needs to be an ε-transition. If this p-fan and the anchor of $E \Rightarrow_{\mathrm{i}} E_2$ overlap only in states adjacent to p but not p itself, p "remains" the center of a fan in E_2, although some of its adjacent states may be merged. This holds regardless of the exact nature of $E \Rightarrow_{\mathrm{i}} E_2$. On the other



(a) the center of the fan lies outside the cycle



(b) the center of the fan lies on the cycle

Figure 4.5: Showcase examples for the convergence of cycle and fan elimination, accompanying Lem. 4.2.5 (ε-labels are omitted).

hand, for this kind of overlap neither an expansion anchor nor a second fan are affected by $E \Rightarrow_{\triangleleft} E_1$. We find

$$E \Rightarrow_{\triangleleft[p]} E_1 \Rightarrow_{\mathrm{i}} E_3 \text{ and } E \Rightarrow_{\mathrm{i}} E_2 \Rightarrow_{\triangleleft[p]} E_3$$

for this case easily.

So assume that the overlap of the fan and the i-anchor contains at least one transition. Since p is incident to each transition of the fan, the overlap includes p as well. We distinguish by the rewriting from E to E_2.

- Suppose $E \Rightarrow E_2$ is the expansion $E \Rightarrow_{[t]} E_2$. Since t is incident to p by assumption, and t carries a compound label, t is one of the transitions leaving p, say, $t = t_1 = (p, r_1, p_1)$. After fan elimination, E_1 contains the transition $t_1' = (p', r_1, q_1)$, which allows for the same type of expansion as t does in E. Let E_3 be the EFA that satisfies $E_1 \Rightarrow_{[t_1']} E_3$.

  On the other hand, expanding t_1 in E removes t_1 and possibly replaces it with elements that connect p to q_1. As the in-degree of p is not increased, E_2 contains a fan with center p, too. Eliminating this fan in E_2 removes p and t_ε, and replaces the $t_i = (p, r_i, q_i)$ with $t_i' = (p', r_i, q_i)$ for $i \neq 1$. Of the transitions introduced by expanding t_1 in E, those which leave p in E_2 are reconnected to leave p'. This yields the EFA which is otherwise obtained by expanding t_1' in E_1, namely E_3. We have

  $$E \Rightarrow_{\triangleleft[p]} E_1 \Rightarrow_{[t_1']} E_3 \text{ and } E \Rightarrow_{[t_1]} E_2 \Rightarrow_{\triangleleft[p]} E_3,$$

  which completes this case. An example where $\Rightarrow_{[t_1]}$ is a star expansion is shown in Fig. 4.6.

- If the rewriting from E to E_2 is a fan elimination, too, let $E \Rightarrow_{\triangleleft[q]} E_2$ for some $q \neq p$. Since we assume that the two fans share a transition, their centers are adjacent. This gives rise to three structurally distinct overlaps of two fans, as sketched in Fig. 4.7. We address them separately in the following:

  - Both fans are in-fans and the ε-transition into q leaves p (Fig. 4.7a). The transitions leaving q will be denoted $t_i' = (q, s_i, q_i)$, which also gives the denotation of further states adjacent to q. Observe that $E \Rightarrow_{\triangleleft[p]} E_1$ leaves a fan with center q in E_1, while $E \Rightarrow_{\triangleleft[q]} E_1$ leaves one with center p in E_2. Eliminating the remaining fan in either EFA yields an EFA with transitions among p', the p_i, and the q_j as follows: for each p_i, $i \geq 2$, there is a transition (p', r_i, q_i) and for each q_j there is a transition (p', s_j, q_j). So the relative order of eliminations is irrelevant, i.e., we find

    $$E \Rightarrow_{\triangleleft[p]} E_1 \Rightarrow_{\triangleleft[q]} E_3 \text{ and } E \Rightarrow_{\triangleleft[q]} E_2 \Rightarrow_{\triangleleft[p]} E_3.$$

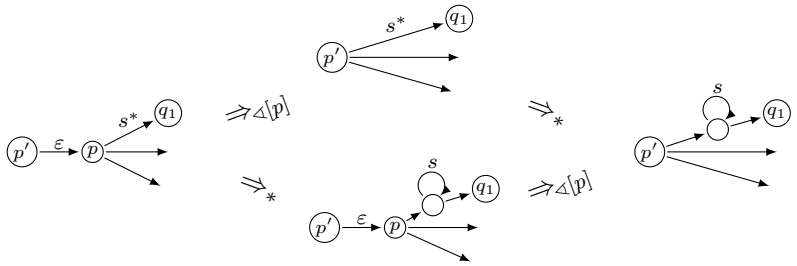


Figure 4.6: Divergence resulting from expansion and fan elimination, and its convergence (cf. first case of Lem 4.2.6)

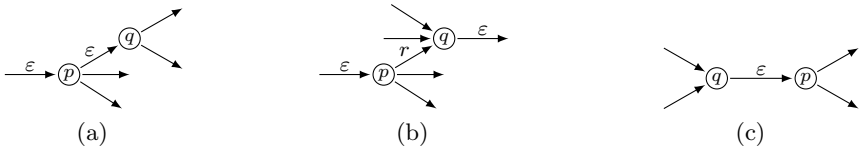


Figure 4.7: Overlaps of fans with adjacent centers p and q (cf. second case of Lem. 4.2.6)

- The fan with center q is an out-fan and there are transitions from p to q (Fig. 4.7b). Let q' denote the state dominated by q, i.e., let (q, ε, q') be the sole transition leaving q. Eliminating both fans in succession, in any order, makes transitions (p, r_j, q) of the overlap into transitions (p', r_j, q'). As these are the only transitions relevant for convergence, the claim follows for this case.

- The fan with center q is an out-fan and the ε-transition into p leaves from q (Fig. 4.7c). In this case, E_1 and E_2 differ only in the name of one state: q in E_2 resp. p in E_1. We thus find $E_1 \cong E_2$, i.e., structurally there is no actual divergence. This becomes obvious from the figure and is not treated formally.

Any further overlap of fans, resp. a possibly resulting divergence, is symmetric to one of above cases. Thus this case is complete.

This completes the proof. □

Theorem 4.2.7. *The ARS $\mathfrak{B}$ is locally confluent, and the normal form of any EFA in $\mathfrak{B}$ is unique.*

Proof. Given any EFA E, suppose $E \Rightarrow_{\mathfrak{B}} E_1$ and $E \Rightarrow_{\mathfrak{B}} E_2$. Any combination of rewritings from E to E_1 and E_2 is within the domain of one of Lems. 4.2.4, 4.2.5, and 4.2.6. Thus follows $E_1 \sim_{\mathfrak{B}} E_2$, so $\mathfrak{B}$ is locally confluent. Since $\mathfrak{B}$ is terminating, it follows from Newman's Lemma that the normal forms in $\mathfrak{B}$ are unique. □

We will get back to the normal form of A_r^0 in $\mathfrak{B}$ at several times. Notice that such a normal form is a finite automaton already, i.e., $\mathfrak{B}$ converts an expression r — in the form of A_r^0 — to a unique FA. In the following, we denote this FA as $A_r^{\mathfrak{B}}$.

The local confluence of $\Rightarrow_{\mathfrak{B}}$ implies that $\sim_{\mathfrak{B}}$ is an equivalence relation. This property will prove particularly useful when establishing further results of our construction.

Proposition 4.2.8. *The relation $\sim_{\mathfrak{B}}$ is an equivalence on* **EFA**.

Proof. Reflexivity and symmetry of $\sim_{\mathfrak{B}}$ follow immediately. To show transitivity, assume $E_1 \sim_{\mathfrak{B}} E_2$ and $E_2 \sim_{\mathfrak{B}} E_3$. This means that there are EFAs E_{12} and E_{23} with rewriting sequences $E_1 \Rightarrow_{\mathfrak{B}}^{\star} E_{12}$ and $E_2 \Rightarrow_{\mathfrak{B}}^{\star} E_{12}$, as well as $E_2 \Rightarrow_{\mathfrak{B}}^{\star} E_{23}$ and $E_3 \Rightarrow_{\mathfrak{B}}^{\star} E_{23}$. So the pair (E_{12}, E_{23}) is a divergence of E_2 in $\mathfrak{B}$; but since $\Rightarrow_{\mathfrak{B}}$ is locally confluent, there is some E_{13} satisfying $E_{12} \Rightarrow_{\mathfrak{B}}^{\star} E_{13}$ and $E_{23} \Rightarrow_{\mathfrak{B}}^{\star} E_{13}$. Thus follows $E_1 \sim_{\mathfrak{B}} E_3$, so $\sim_{\mathfrak{B}}$ is transitive, too. □

We proceed with showing local confluence $\mathfrak{X}$ and $\mathfrak{Y}$, which is straightforward in comparison. As a matter of fact, the restrictions **(X1)**, **(X2)**, **(Y1)**, and **(Y2)**, were devised for the very purpose of enforcing confluence, resp. unique outputs, of the full construction.

Lemma 4.2.9. *The ARS $\mathfrak{X}$ is locally confluent.*

Proof. We show that $A \Rrightarrow_{X[p]} A_1$ and $A \Rrightarrow_{X[q]} A_2$ implies $A_1 \Rrightarrow_{X[q]} A_3$ and $A_2 \Rrightarrow_{X[p]} A_3$ for some FA A_3. If the $X[p]$- and the $X[q]$-anchor share a state that is neither p nor q, then A_1 contains an $X[q]$-anchor and A_2 contains an $X[q]$-anchor, so X-elimination rewrites either FA to A_3 as stated. On the other hand, if one of p or q is part of the overlap, A contains a transition either from p to q or vice versa. But then one of the eliminations violates **(X2)**, contrary to the assumption that both eliminations are valid. □

Lemma 4.2.10. *The ARS $\mathfrak{Y}$ is locally confluent.*

Proof. Assuming $A \Rrightarrow_{Y[p]} A_1$ and $A \Rrightarrow_{Y[q]} A_2$, the proof proceeds similarly to that of Lem. 4.2.9, and is not given in detail. Again, it follows that the overlap of the two anchors is either trivial, which yields the claim following Prop. 4.2.3, or that one elimination does not respect **(Y2)**, contrary to the initial assumption. □

We have arrived at the main result of this section, namely, that the ARS $\mathfrak{C}$, while respecting the meta rules **(X1)**, **(X2)**, **(Y1)**, and **(Y2)** admits normal forms. In other words, this modified ARS manages the conversion of regular expressions to uniquely determined finite automata.

Theorem 4.2.11. *Let $A_r^0 \Rightarrow^\star A_r$, where A_r is a normal form of $\mathfrak{C}$. If all steps of this rewriting are valid, then A_r is unique.*

Proof. It follows from Cor. 4.2.2 and the notation introduced in this section, that the rewriting of A_r^0 to A_r can be written as

$$A_r^0 \Rightarrow_{\mathfrak{B}}^\star A_1 \Rrightarrow_X^\star A_2 \Rrightarrow_Y^\star A_r.$$

We have shown that each of the involved ARSs $\mathfrak{B}$, $\mathfrak{X}$, and $\mathfrak{Y}$ admits unique normal forms. Therefore the intermediate FAs A_1 and A_2, as well as the resulting FA A_r, are unique. □

4.3 Invariance under Strong Star Normal Form

As mentioned in Ch. 3, the conventional star normal form was originally devised as a preprocessing step in the construction of position automata. An expression and its star normal form provide the same position automaton; however, for arbitrary expressions the construction might require cubic time in the input size, while quadratic running time is guaranteed for expressions in normal form [8]. From a different perspective, one might argue that the star normal form is implicitly computed upon constructing the position automaton.

We claim that in analogy to the above, our construction is invariant with respect to SSNF, i.e., that $A_r = A_{r^\bullet}$ holds for every expression r. In the following we show that this equality is already realized in the basic ARS $\mathfrak{B}$, i.e., we show $A_r^{\mathfrak{B}} = A_{r^\bullet}^{\mathfrak{B}}$.

To this end, we introduce some additional notation: Given a transition $t = (p, r, q)$ we set $t^\circ := (p, r^\circ, q)$ and $t^\bullet := (p, r^\bullet, q)$. Moreover, the EFA derived from E by replacing t with t°, resp. $t^\bullet$, will be denoted $E[t^\circ/t]$, resp. $E[t^\bullet/t]$.

The following lemma lies at the heart of proving our claim.

Lemma 4.3.1. *Let t be a loop transition of E. Then $E \sim_{\mathfrak{B}} E[t^\circ/t]$.*

Proof. Let $E \in \mathbf{EFA}$ be arbitrary and let $t = (p, r, p)$ be a loop of E. We prove the statement by induction on the size of r. For $r \in \mathcal{A} \cup \{\varnothing\}$ we have $r^\circ = r$ by definition, thus also $t^\circ = t$, and the claim follows trivially. For $r = \varepsilon$, we observe that t forms an ε-cycle already: eliminating this cycle coincides with removing the transition. On the other hand, since $\varepsilon^\circ = \varnothing$, the EFA $E[t^\circ/t]$ can be zero expanded in t°. Either conversion removes the loop t, resp. t°, so E and $E[t^\circ/t]$ converge in $\mathfrak{B}$.

Assume that the claim holds for expressions strictly smaller than r and distinguish by the structure of r.

- $r = st$: If st is not nullable, the inductive step is trivial due to $(st)^\circ = st$; this yields $E[t^\circ/t] = E$ immediately. Otherwise, we consider the full expansion of t. Denote this rewriting as

  $$E \Rightarrow . \, E' \Rightarrow_{\mathrm{ex}}^{\star} E'',$$

  where the first step — product expansion — replaces $t = (p, st, p)$ with a new state q and transitions (p, s, q) and (q, t, p). Since E'' is derived from E' by expansions only, E'' still contains p and q as states. Moreover, since with st both s and t are nullable, E'' contains ε-paths from p to q and from q to p, so p and q lie on a common ε-cycle. We consider the corresponding cycle elimination,

  $$E'' \Rightarrow_{\circlearrowleft} E''',$$

  as a two-step process, as follows: the first step merges p and q, yielding an intermediate "volatile" EFA $E''_{[p=q]}$. The second step merges the remaining states of the original cycle with the merge state of p and q and removes the ε-transitions of the original cycle. Observe, however, that $E''_{[p=q]}$ rewrites to E''' via two successive cycle eliminations: merging p and q in E'' transforms the ε-path from p to q and that from q to p into a separate ε-cycle each; this is shown in upper part of Fig. 4.8.

  Now consider the EFA $E[t^\circ/t]$ where the label st of t is replaced with $(st)^\circ = s^\circ + t^\circ$. Sum expanding $E[t^\circ/t]$ in t° yields an EFA F with two loops,

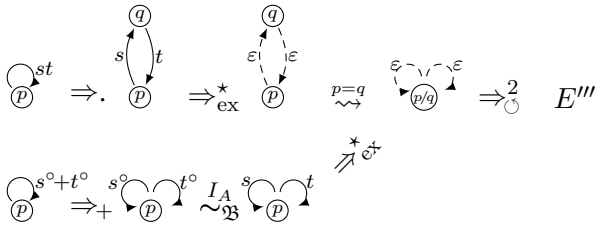


Figure 4.8: First case in the proof of Lem. 4.3.1: the loop-label st, which is nullable, is replaced with $(st)^\circ = s^\circ + t^\circ$.

$t_1{}^\circ = (p, s^\circ, p)$ and $t_2{}^\circ = (p, t^\circ, p)$. The inductive hypothesis applies to loops labeled s and t, it can be applied "backwards" to the t_i. This gets us the EFA $F \sim_{\mathfrak{B}} F[t_1/t_1{}^\circ][t_2/t_2{}^\circ]$. Full expansion of both t_1 and t_2 in $F[t_1/t_1{}^\circ][t_2/t_2{}^\circ]$ yields F', which contains two ε-cycles, since s and t are nullable. This is the EFA we considered as an intermediate step of $E'' \Rightarrow_{\circlearrowleft} E'''$ before. Eliminating the two ε-cycles that emerged from expanding t_1 and t_2 yields E''', too (shown in the lower part of Fig. 4.8).

Altogether we have shown

$$E \Rightarrow^\star E''',\ E[t^\circ/t] \sim_{\mathfrak{B}} F[t_1/t_1{}^\circ][t_2/t_2{}^\circ],\ \text{and}\ F[t_1/t_1{}^\circ][t_2/t_2{}^\circ] \Rightarrow^\star E''',$$

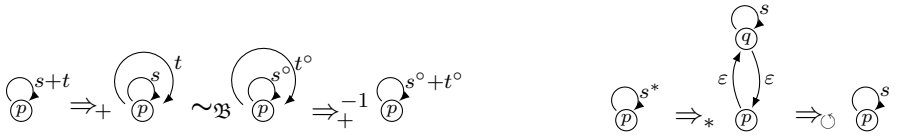
so $E \sim_{\mathfrak{B}} E[t^\circ/t]$ follows.

- $r = s + t$: Sum expansion replaces t with loops $t_1 = (p, s, p)$ and $t_2 = (p, t, p)$, yielding the EFA E'. The inductive assumption applies to both t_1 and t_2 in E', i.e., $E' \sim_{\mathfrak{B}} E'[t_1{}^\circ/t_1][t_2{}^\circ/t_2]$. However, the parallel loops s° and t° also emerge from expanding $t^\circ = (p, (s + t)^\circ, p) = (p, s^\circ + t^\circ, p)$ in $E[t^\circ/t]$ (Fig. 4.9a). Thus $E \sim_{\mathfrak{B}} E[t^\circ/t]$ follows for this case, too.

- $r = s^*$: Star expansion $E \Rightarrow_{[t]} E'$ removes t and adds a new state q along with transitions (p, ε, q), (q, s, q), and (q, ε, p). Here, the first and third transition form an ε-cycle. Eliminating this cycle in E' yields an EFA which is identical (up to the name of a single state) to $E[t^\circ/t]$ (Fig. 4.9b). Notice that $E \sim_{\mathfrak{B}} E[t^\circ/t]$ holds because $E[t^\circ/t]$ is directly constructed from E, i.e., the inductive assumption is not used in this case.

□

Theorem 4.3.2. *Let t be any transition of E. Then $E \sim_{\mathfrak{B}} E[t^\bullet/t]$.*



(a) sum, where $(s+t)^\circ = s^\circ + t^\circ$ (b) iteration, where $s^{*\circ} = s$

Figure 4.9: Second and third case in the proof of Lem. 4.3.1

Proof. For $t = (p, r, q)$ we prove the claim by induction on $|r|$. The base case, $r \in \mathcal{A} \cup \{\varepsilon, \varnothing\}$ holds trivially by definition of $^\bullet$. Suppose the claim is true for labels smaller than r and distinguish by the structure of r:

- $r = st$: Product expansion $E \Rightarrow_{[t]} E'$ replaces t with a new state x and transitions $t_1 = (p, s, x)$ and $t_2 = (x, t, q)$. The inductive assumption applies to t_1 and t_2, thus $E' \sim_{\mathfrak{B}} E'[{t_1}^\bullet/t_1][{t_2}^\bullet/t_2]$. On the other hand we find $E[t^\bullet/t] \Rightarrow_{[t^\bullet]} E'[{t_1}^\bullet/t_1][{t_2}^\bullet/t_2]$, so E and $E[t^\bullet/t]$ converge in $\mathfrak{B}$.
- $r = s + t$: As in the previous case except for the obvious modifications.
- $r = s^*$: Upon star expansion $E \Rightarrow_{[t]} E'$, the transition t is removed while a new state x and transitions $t_1 = (p, \varepsilon, x)$, $t_2 = (x, s, x)$, and $t_3 = (x, \varepsilon, q)$ are added. As t_2 is a loop, Lem. 4.3.1 implies $E' \sim_{\mathfrak{B}} E'[{t_2}^\circ/t_2]$. Recall that Prop. 3.2.1 states $|s^\circ| \leq |s|$; so the inductive assumption applies to ${t_2}^\circ$, and yields $E' \sim_{\mathfrak{B}} E'[{t_2}^{\circ\bullet}/t_2]$. We thus also have $E \sim_{\mathfrak{B}} E'[{t_2}^{\circ\bullet}/t_2]$. Now consider $E[t^\bullet/t]$, where $t^\bullet = (p, s^{*\bullet}, q) = (p, s^{\circ\bullet *}, q)$. The expansion $E[t^\bullet/t] \Rightarrow_{[t^\bullet]} E''$, star expansion, replaces $t^\bullet$ with a state x' and transitions $t'_1 = (p, \varepsilon, x')$, $t'_2 = (x, s^{\circ\bullet}, x')$, and $t'_3 = (x', \varepsilon, q)$. A simple renaming shows that E'' equals $E'[t^{\circ\bullet}/t]$, so in conclusion, we find $E \sim_{\mathfrak{B}} E[t^\bullet/t]$.

In each case, replacing t with $t^\bullet$, i.e., putting the label of any transition in SSNF, yields convergent EFAs in $\mathfrak{B}$. This completes the proof. □

The initial claim of this section — invariance of the FA yielded by our construction under taking the SSNF of the input expression — now follows by applying Thm. 4.3.2 to the primal EFA A^0_r.

Corollary 4.3.3. *For any expression r we find $A_r = A_{r^\bullet}$.*

Proof. By virtue of Thm. 4.3.2 we find $A^0_r \sim_{\mathfrak{B}} A^0_{r^\bullet}$, since $A^0_{r^\bullet} = A^0_r[t^\bullet/t]$ for the sole arc t in A^0_r. This implies $A^{\mathfrak{B}}_r = A^{\mathfrak{B}}_{r^\bullet}$, and this $\mathfrak{B}$-normal FA is converted

in $\mathfrak{X}$ to a unique $\mathfrak{X}$-normal FA, which in turn is converted in $\mathfrak{Y}$ to a unique $\mathfrak{Y}$-normal FA. □

4.4 Implementation Details and Running Time

The canonical implementation simply puts the ARSs $\mathfrak{B}$, $\mathfrak{X}$, and $\mathfrak{Y}$, which constitute our construction, "straight into code". That is, each ARS is realized by a main loop wherein some data structure that represents the intermediate EFAs A_r^k, is repeatedly modified by valid reductions until no rule applies anymore.

In the case of $\mathfrak{B}$, we may repeatedly choose any transition labeled $\varnothing$or by some compound expression. These transitions can be kept, e.g. in a list which is updated with each rewriting step. Whenever a new ε-transition is introduced, we test whether this transition is part of an ε-cycle or a fan, and apply the according elimination. As the first appearance of an elimination anchor is right after an expansion, this captures all eliminations that may possibly occur in a rewriting. Since $\mathfrak{B}$ is confluent, this strategy certainly yields $A_r^{\mathfrak{B}}$. For $\mathfrak{X}$ and $\mathfrak{Y}$, a similar approach may be used, while additionally respecting the meta-rules **(X1)**, **(X2)** and **(Y1)**, **(Y2)**. This method invariably terminates in A_r, solely due to local confluence of the three rewriting systems.

We present a more subtle approach which produces FA with useful states only, runs faster — not asymptotically, though — and is less complex. For compound expressions that contain $\varnothing$, the proposed modifications yield FAs that are smaller than those obtained with original rewriting system.

The key to an improved implementation is to avoid certain rewritings altogether, of course without jeopardizing the output. This will be realized by adding preprocessing steps that manipulate the input expressions in a way that makes most eliminations superfluous and keeps the intermediate EFAs small. In effect, some workload is transferred from rewriting EFAs to rewriting expressions, which, arguably, is easier to implement.

We observe that the FA A_r is not necessarily trim. This is ultimately a consequence of the syntax we allow for expressions, namely, occurrences of symbol $\varnothing$. Consider for instance the expression $r = a + \varnothing b$, which denotes $\mathrm{L}(r) = \{a\}$. Our construction converts r into the non-trim FA that is shown in Fig. 4.10a, instead of the smaller and trim FA shown in Fig. 4.10b.

(a) (b)

Figure 4.10: FAs constructed from $a + \varnothing b$ and with the useless state removed.

Of course, we could enrich our construction with rules that remove useless states. This can be done on a local level by deleting any non-initial state with in-degree zero, resp. any non-final states with out-degree zero. However, showing that the enriched ARS produces unique outputs and is invariant under SSNF, as we did for our construction in the previous sections, requires to consider ever more (sub)cases than those which were necessary in the respective proofs already. Instead, we realize this additional feature on the expression level, by preventing zero expansion altogether, i.e., by removing all instances of $\varnothing$. Being at that, we also remove all occurrences of ε; this reduces the number of fan eliminations that happen in the conversion.

Definition 7. An expression is $\varnothing$-*reduced* if it satisfies either $r = \varnothing$ or $|r|_\varnothing = 0$. Likewise, an expression is ε-*reduced* if either $r = \varepsilon$ or $|r|_\varepsilon = 0$ holds. An expression that is both $\varnothing$-reduced and ε-reduced is *reduced*.

Any expression can be converted into an equivalent reduced expression in linear time. This is realized by applying the following two-step procedure in a bottom-up fashion to the parse tree of r.

1. Remove all occurrences of $\varnothing$ as proper subexpressions: replace $\varnothing s$ and $s\varnothing$ with $\varnothing$. Replace $\varnothing + s$ and $s + \varnothing$ with s. Replace $\varnothing^*$ and $\varnothing^?$ with ε.
2. Remove all occurrences of ε as proper subexpressions: replace εs and $s\varepsilon$ with s. Replace $\varepsilon + s$ and $s + \varepsilon$ with $s^?$. Replace ε^* and $\varepsilon^?$ with ε.

It should be fairly obvious that these rules convert an expression into an equivalent reduced expression. Notice that $\varnothing$-reduction needs to be carried out first, since $\varnothing$-reduction might introduce instances of ε, while ε-reduction does not introduce $\varnothing$. Let us further mention without proof that the resulting expression is uniquely determined. Given the expression r, we denote its reduced equivalent that results from above conversions with $\mathrm{red}(r)$.

If an expression is reduced, there is clearly no need to implement $\varnothing$-expansion. More importantly, each $A^k_{\mathrm{red}(r)}$ is trim, as follows by straightforward induction on k. The rewriting rules of $\mathfrak{C}$ ensure that each A^k_r is normalized for arbitrary r; so for reduced expressions, our construction produces trim normalized automata.

Proposition 4.4.1. *For any expression r we find $A_{\mathrm{red}(r)} = \mathrm{trim}(A_r)$.*

Proof. It is sufficient to prove $A^{\mathfrak{B}}_{\mathrm{red}(r)} = \mathrm{trim}(A^{\mathfrak{B}}_r)$, the claim then follows from confluence of $\mathfrak{X}$ and $\mathfrak{Y}$. We first show how, resp. which, steps in the process of reducing an expression are realized by the rules in $\mathfrak{B}$.

First, let $\varnothing^* \in \mathrm{sub}(r)$, then, at some point, a transition $(p, \varnothing^*, q)$ is present in some A^k_r. Zero-elimination replaces this transition with (p, ε, z), $(z, \varnothing, z)$, and (z, ε, q), where z is new state. Now null-expansion removes the loop transition, and fan elimination replaces the two ε-transitions with one, namely (p, ε, q). In effect, the transition $(p, \varnothing^*, q)$ was replaced by (p, ε, q), which is the same replacement, label-wise, as with $\varnothing$-reducing the expression.

Second, let $s + \varnothing \in \mathrm{sub}(r)$: the corresponding transition $(p, s + \varnothing, q)$ is replaced with (p, s, q) and $(p, \varnothing, q)$ by sum expansion, and the $\varnothing$-transition is removed by zero expansion. So $(p, s + \varnothing, q)$ is replaced by (p, s, q), as in the process of reducing r by first reducing its subexpressions. Recall that we treat an option as a special case of a sum. Hence the transition $(p, \varnothing^?, q)$, which is treated as $(p, \varnothing + \varepsilon, q)$, is replaced with (p, ε, q), as in the step of the reduction of r.

In the same fashion, it is straightforward to see that the removal of ε as a factor is realized by fan-elimination, and that replacing ε^* with ε is realized by cycle elimination. Replacing $s + \varepsilon$ with $s^?$ makes no difference in $\mathfrak{B}$, since we already treat options as sums.

It remains to consider $\varnothing$ as a factor; for any other occurrence of $\varnothing$ (or ε) as a subexpression, $\mathfrak{B}$ implicitly realizes the steps in reducing the input expressions. So let r be an expression that is free of ε and where every instance of $\varnothing$ is a base. Consider a transition $(p, \varnothing s, q)$, which is replaced with $(p, \varnothing, z)$ and (z, s, q) by product expansion. Next, null expansion removes $(p, \varnothing, z)$, to the effect that z is unreachable in A_r, if z is a state of A_r in the first place. The same follows for any state that is introduced by further expanding (z, s, q). These states, including z, are removed by "trimming" A_r. The same is achieved by reducing r, where $\varnothing s$ is replaced with $\varnothing$, and the transition $(p, \varnothing, q)$ is removed through null expansion. The argument is symmetric for $s\varnothing$, where reasoning is done wrt. co-reachability.

Conversely, any state that is unreachable (co-unreachable) in A_r must have emerged by expanding a transition labeled with a subexpression of $\varnothing s$ ($s\varnothing$). The only rule that affects connectivity appears in $\mathfrak{B}$, and we agreed that all occurrences of n in r are factors. □

As we have shown in (the proof of) Prop. 4.4.1, several features of a reduced expression are realized in $\mathfrak{B}$ on the FA level. This is similar to the result given in Cor. 4.3.3, where we showed that $\mathfrak{B}$ is invariant under SSNF, or equivalently, that the SSNF of the input expression is implicitly computed upon rewriting. We use this fact by feeding $r^\bullet$ to the rewriting system; our findings imply that this does not change the resulting FA. In fact, the overall number of rewritings might decrease with this preprocessing, since $^\bullet$ is monotonic, i.e., $r^\bullet$ might contain fewer unary operators than r, but certainly not more (Prop. 3.2.1).

However, there is a much more important reason for this preprocessing step: if an expression is in SSNF, no ε-cycles emerge in the course of the construction. To see this, we first prove a more general property. Recall that $L_E(p, q)$ denotes the set of words that carry the state p to the state q in an EFA E.

Lemma 4.4.2. *If r is in* SSNF *and q is a state of A_r^n, then $\varepsilon \notin L_{A_r^n}(q, q)$.*

Proof. We prove the claim by induction on n.

Clearly, the claim holds for A_r^0. Assume the claim holds for A_r^i and consider any A_r^{i+1}. Formally, we need to distinguish by the rewriting from A_r^i to A_r^{i+1}. However, for every state q of A_r^{i+1} that is also a state of A_r^i, the statement follows immediately, since in that case $L_{A_r^{i+1}}(q, q) = L_{A_r^i}(q, q)$ follows from soundness of the conversions. The two conversions that introduce new states are product expansion and star expansion; we consider only these rewritings in detail.

Assume $A_r^i \Rightarrow. A_r^{i+1}$, where q is introduced by expanding (p, st, p'). If st is not nullable, then at least one out of s and t is not nullable, either. This implies $\varepsilon \notin L_{A_r^{i+1}}(q', q)$, or $\varepsilon \notin L_{A_r^{i+1}}(q, q')$ for arbitrary q', thus in particular $\varepsilon \notin L_{A_r^{i+1}}(q, q)$. If st is nullable, then $\varepsilon \in L_{A_r^i}(p, p')$; in that case, the inductive assumption implies $\varepsilon \notin L_{A_r^i}(p', p)$. As product expansion is sound, $\varepsilon \notin L_{A_r^{i+1}}(p', p)$ follows, too. Since q is only reached through p and q only reaches other states through p', $\varepsilon \in L_{A_r^{i+1}}(q, q)$ requires $\varepsilon \in L_{A_r^{i+1}}(p', p)$, which was just disproved. Thus follows $\varepsilon \notin L_{A_r^{i+1}}(q, q)$ for this case.

Now assume $A_r^i \Rightarrow_* A_r^{i+1}$, where q is introduced by expanding (p, s^*, p'). That is, A_r^{i+1} contains transitions (p, ε, q), (q, s, q), and (q, ε, p'). Since the claim assumes that r is in SSNF, Thm. 3.2.8 implies $\varepsilon \notin \mathrm{L}(s)$, so, given the transitions in A_r^{i+1}, $\varepsilon \in L(q, q)$ requires $\varepsilon \in L_{A_r^{i+1}}(p', p)$. As p' and p are also states in A_r^i, this further implies $\varepsilon \in L_{A_r^i}(p', p)$. However, we certainly have $\varepsilon \in L_{A_r^i}(p, p')$, because of the transition (p, s^*, p'), so this last implications contradicts the inductive assumption. □

Corollary 4.4.3. *If r is in* SSNF*, then A_r^n is free of ε-cycles for $n \in \mathbb{N}$.*

Proof. The property $\varepsilon \in L_E(q, q)$, holds for every state q on an ε-cycle in E. According to Lem. 4.4.2, no such state is present in A_r^n if r is in SSNF. Consequently, A_r^n is free of ε-cycles. □

Therefore, if we replace an input expression with its SSNF, there is no need to implement ε-cycle elimination. Manipulating (the parse of) an expression is arguably easier to realize than eliminating ε-cycles in an EFA. Nevertheless, the latter is still possible in linear time, as is discussed by Ilie & Yu [30]. We finally show that putting an expression in SSNF does not accidentally nullify the advantages we obtained by reducing the expression in the first preprocessing step.

Proposition 4.4.4. *If r is reduced, then $r^\bullet$ is also reduced.*

Proof. Let r be reduced, then $r = \varnothing$, $r = \varepsilon$, or $|r|_\varnothing = |r|_\varepsilon = 0$. In the first and second case, the definition of $^\bullet$ yields $r^\bullet = \varnothing^\bullet = \varnothing$, resp. $r^\bullet = \varepsilon^\bullet = \varepsilon$, so $r^\bullet$ is reduced in either case. In the third case, we find that whenever $\varnothing$ or ε is produced upon computing $r^\bullet$, an instance of either symbol is required in the first place (cf. Def. 2). Thus follows the claim. □

These preprocessing steps allow to tweak an implementation of $\mathfrak{B}$ even more, by simplifying fan elimination. Consider a step in the conversion of an expression that is reduced an in SSNF, i.e., some r with $r = \mathrm{red}(r)^\bullet$. Since r is reduced, it contains no instances of ε. Assume now that A_r^k does not contain any fan, and that a fan appears with the rewriting $A_r^k \Rightarrow A_r^{k+1}$ in A_r^{k+1}. This implies that an ε-transition is introduced by this rewriting. The only conversions that produce ε-transitions are sum expansion, for a transition $(p, s^?, q)$[2], and star expansion, for a transition (p, s^*, q). For sum expansion, however, the produced ε-transition is parallel to a second transition, so it yields neither an out-fan with center p, nor an in-fan with center q. It follows that the only rewriting that might introduce a fan is a star expansion.

More precisely, we find that the ε-transition of a fan is one of the two ε-transitions that are produced with star expansion. To guarantee that an FA is $\Rightarrow_\triangleleft$-normal, it is sufficient to eliminate any fan right after its creation. Here, we apply fan elimination right after star expansion, if a fan arises. The advantage

[2] recall that we treat an option as a particular kind of sum

of this approach is that we identify a fan locally, i.e., we avoid searching the whole EFA. We employ this strategy in Alg. 1.

A variation of this approach is to use several variants of star expansion which "include" follow-up fan elimination(s). This requires five clones of the expansion we use, each managing a different possibility where a fan may appear upon expansion. Expanding the transition $t = (p, s^*, q)$ might produce no fan at all, or exactly one, centered in either incident state, or two fans. A further rule replaces the label s^* with s, if $p = q$. An ARS using these rules was investigated in [22]; except for the effects of Y-elimination, it produces the same FAs as the ARS we consider.

Next, we consider implementation aspects concerning X- and Y-eliminations. The details are by and far identical for the two conversions, so we elaborate on X-elimination and argue by analogy for Y.

As we have shown in Lem. 4.2.1, once we have constructed $A_r^{\mathfrak{B}}$, we can apply valid X-elimination until the normal form $A_r^{\mathfrak{X}}$ is reached. This is done in a second loop in Alg. 1. In this loop we first gather all X-anchors of $A_r^{\mathfrak{B}}$. Recall that q is the center of an X-anchor if $\mathrm{d}^-(q) = \mathrm{d}^+(q) = 2$, and all in- and out-transitions of q are ε-transitions. These states are found by scanning the set of states. We find that it is sufficient to do this once, since no new X-anchors, resp. centers, are introduced with valid X-elimination. In other words, any X-anchor present in an FA after X-elimination is also present in the FA before elimination.

Proposition 4.4.5. *Let $E \Rrightarrow_X E'$ and let q be the center of an X-anchor in E'. Then q is the center of an X-anchor in E.*

Proof. By the nature of X-elimination, every state in E' is also a state in E. If p and q are not adjacent in E, the claim follows, since $X[p]$-elimination does not alter the transitions incident to q, i.e., these agree in E and E'.

Suppose otherwise, then E contains the transition (p, ε, q) (or (q, ε, p), which is symmetric), since p is incident to ε-transitions only. With X-elimination in p, the in-degree of q increases by one, i.e., $\mathrm{d}^-_{E'}(q) = \mathrm{d}^-_E(q) + 1$. But E' contains an $X[q]$-anchor by assumption, which implies that $\mathrm{d}^-_{E'}(q) = 2$, so we find $\mathrm{d}^-_E(q) = 1$. Thus the only transition entering q in E is an ε-transition, meaning that q is the center of a fan. This contradicts the assumption that $X[p]$-elimination is valid, so p and q are not adjacent. □

Notice, however, that X-elimination can "destroy" adjacent X-anchors, i.e., the center of an X in E is not necessarily the center of an X in E', if $E \Rightarrow_X E'$ is assumed. Thus, if we create a list of all X-anchors of $A_r^{\mathfrak{B}}$, resp. their central states, and choose states from this list throughout the rewriting to locate elimination anchors, we need to check in each case whether the chosen state is "still" the center of an anchor in the current automaton.

In the following, let Q_X denote the states in $A_r^{\mathfrak{B}}$ that are the center of an X-anchor each. Consider the meta-rules that lead to valid X-elimination, given in Def. 6. The rule **(X2)** introduces elimination priorities among X-anchors. As we have already argued, there are no cyclic preferences wrt. **(X2)**. Interpreted differently, the set of center states and the ε-transitions among theses states form an acyclic subgraph of $A_r^{\mathfrak{B}}$. Therefore, the domination relation induces a topological order $\prec_X$ on Q_X, and an $X[q]$-anchor can be eliminated iff it is $\prec_X$-minimal in Q_X. The set Q_X can be ordered in linear time by standard algorithms (e.g. , as given in [12]). In a loop, we then choose the minimal element of this order and, if the state is still the center of an X-anchor, apply elimination.

Each step discussed for (valid) X-elimination above holds for Y-elimination with the obvious changes. Similar to the denotation for X-elimination, Q_Y contains the center states of Y-anchors in $A_r^{\mathfrak{X}}$, and $\prec_Y$ is a topological order that is consistent with **(Y2)**.

Theorem 4.4.6. *The FA* $\mathrm{trim}(A_r)$ *is constructed by Alg. 1 in time* $\mathcal{O}(|r|)$.

Proof. Either preprocessing step — reducing the input expression and computing the SSNF— can be realized in linear time by a bottom-up term rewriting system. So let r be reduced and in SSNF. We assume that A in Alg. 1 is represented by a dynamic graph with labeled arcs[3]. We separately consider the three loops in Alg. 1, which realize $\mathfrak{B}$, $\mathfrak{X}$, and $\mathfrak{Y}$.

To test for the existence of transitions with compound labels, we assume that Alg. 1 keeps a set T with (pointers to) the corresponding arcs of A. This set is initialized with the sole transition (p, r, q), if r is a compound expression, otherwise it remains empty. In each iteration of the loop $\mathfrak{B}$, an arc/transition is removed from T, and A is modified according to the label of this item. If a compound transition is produced, the set T is updated accordingly. This is

[3]This data structure is provided by libraries such as LEDA [35]. Therein, the operations of adding or removing a vertex or an arc to a graph take constant time.

Algorithm 1: Construction of a trim FA from an expression.

Input: Regular expression r
Output: Finite automaton A with $\mathrm{L}(A) = \mathrm{L}(r)$
$r \leftarrow \mathrm{red}(r)^\bullet$
$A \leftarrow (\{q_0, q_f\}, \mathcal{A}_r, (q_0, r, q_f), q_0, q_f)$

$\mathfrak{B}$ **while** A *contains transitions with compound label* **do**
  remove (p, ρ, q), with compound ρ, from A
  switch ρ **do**
    case st
      add state z and transitions (p, s, z), (z, t, q), to A
    case $s + t$
      add transitions (p, s, q), (p, t, q), to A
    case s^*
      add state z and transitions (p, ε, z), (z, s, z), (z, ε, q), to A
      if $\mathrm{d}^+(p) = 1$ **then** apply $\lhd[p]$-elimination
      if $\mathrm{d}^-(q) = 1$ **then** apply $\lhd[q]$-elimination

$\mathfrak{X}$ find $Q_X \subseteq Q_A$ and compute $\prec_X$ on Q_X
while $Q_X \neq \emptyset$ **do**
  $Q_X \leftarrow Q_X \setminus q$ for $\prec_X$-minimal q
  if q *is the center of* X*-anchor* **then** apply $X[q]$-elimination

$\mathfrak{Y}$ find $Q_Y \subseteq Q_A$ and compute $\prec_Y$ on Q_Y
while $Q_Y \neq \emptyset$ **do**
  $Q_Y \leftarrow Q_Y \setminus q$ for $\prec_Y$-minimal $q \in Q_Y$
  if q *is the center of* Y*-anchor* **then** apply $Y[q]$-elimination

done until T is empty. Therefore, no transitions are actually searched in A and the overhead due to T is negligible.

Every compound subexpression of r becomes the label of a transition, which is expanded at some point. Since expansions may not arise otherwise, the number of expansions is $|r|_. + |r|_+ + |r|_*$ (recall that options are treated as sums). It is obvious that each expansion takes $\mathcal{O}(1)$ time, thus the overall time spent with expansions in $\mathfrak{B}$ is in $\mathcal{O}(|r|)$. This does not include the time required for follow-up fan eliminations, which are considered next.

The number of operations necessary for the fan eliminations that might follow immediately after expanding (p, s^*, q), is $\mathcal{O}(\mathrm{d}^-(p) + \mathrm{d}^+(q))$. If this expansion introduces a fan in the first place, we find that s^* is a factor in r: since r is in SSNF, s^* is not a base, and if s^* is an addend, $\mathrm{d}^+(p) > 1$ and $\mathrm{d}^-(q) > 1$ follow. Let t be the factor to the left of s^*, i.e., let $ts^* \in \mathrm{sub}(r)$, and interpret t as a maximal sum of arity k. Then $\mathrm{d}^-(p) = k$ follows, so k transitions need to be "reconnected" for fan elimination in p. After elimination, these transitions are incident to a state that carries a loop and will thus never become the center of a fan itself. The arity of t thus determines the complexity of eliminating an in-fan at most once. Likewise, $\mathrm{d}^+(q) = k'$ if s^* appears in the product s^*t', where t' is a sum of arity k'; again, this value determines the number of operations necessary for eliminating an out-fan with center q. It follows that the number of operations necessary in all fan eliminations together is bounded by arity of all maximal sums in r combined, which is in $\mathcal{O}(|r|)$.

As discussed above, the centers of X-elimination are found by scanning Q_A. Since a state is only introduced with product and star expansion, we find $|Q_A| \leq |r|_. + |r|_*$. So Q_X is found in linear time and is obviously of linear size. Computing a topological order on Q_X that is consistent with domination among anchors takes linear time, too. One iteration of the loop $\mathfrak{X}$ takes constant time, so the time spent for making A X-normal is in $\mathcal{O}(|r|)$, too.

The analysis for Y-elimination parallels that for X-elimination, with the obvious changes. Considering all steps, Alg. 1 constructs $\mathrm{trim}(A_r)$ in time $\mathcal{O}(|r|)$, as claimed. □

5 Conversion Ratio

We continue to investigate the construction presented in Ch. 4: the ARS $\mathfrak{C}$ and its restriction to valid conversions. We have shown two important qualitative properties of FAs that are obtained by this method, namely that these FAs are unique and invariant under taking the SSNF of the input expression. These properties are fundamental for the quantitative analysis we are set about to do in this chapter.

It is natural to ask how the sizes of the involved objects compare. Besides the theoretical main concern of this work — descriptional complexity — this aspect is of practical relevance, as it relates to memory efficiency in applications that employ automata. Considering nowadays availability of cheap main memory, this is not exactly critical for applications on single user machines. The story is different, however, in scenarios where data streams need to be scanned at extremely high rates. This is usually done by dedicated devices, such as routers, which run on specialised hardware and use a more sophisticated, expensive, and hence generally less available type of memory.

The conversion of regular expressions to small finite automata recently gained new attention in the context of network intrusion detection and deep packet inspection. These security measures evaluate the integrity of network connections by scanning the transmitted data. Originally, data streams were scanned for matches from a fixed list of keyword, i.e., an explicitly given set of strings. The flexibility gained from specifying keywords by regular expressions was first pointed out by Sommer & Paxson [50], who proposed a conversion to deterministic finite automata. As we have mentioned in the introductory section of Ch. 3, an exponential blowup cannot be avoided for such automata. On the other hand, once the automaton is present, deterministic automata beat nondeterministic ones wrt. processing time, as the latter kind requires backtracking to guarantee that all branching runs of an input are taken into account.

This led to the suggestion of a hybrid type of automaton, which consists of a deterministic and a nondeterministic "part". A transmission packet is initially processed — i.e., scanned — by the deterministic part of the inspecting

automaton. It is expected that the majority of packets is uncompromised. Thus a random packet is typically rejected almost immediately, as it does not match the filter expression within its first few bytes. The minority of packets is passed to the nondeterministic part of the automaton and processed further. Intuitively, this second part encodes the finer details of the filter, which would consume too much memory in deterministic form. Since the workload is much smaller for this second part, the overhead incurred by backtracking remains manageable. Hybrid automaton apporaches are described, e.g. in [2, 32, 56].

Besides the practical relevance of small automata, we are interested in analyzing the size relationship for its own sake. It is well known that any expression can be converted to such a nondeterministic automaton whose size is linear in that of the expression. This is apparent in the earliest constructions of that type, e.g. those by Ott & Feinstein [41], Kleene [31], and Thompson [51]. These constructions, just like the one presented in Ch. 4, build an automaton based on the structure of an expression: the expression is deconstructed according to its parse tree, and a finite number of states and transitions is introdeuced with each step. The linear "blowup" therein is evident.

In this chapter we bound the ratio of automaton to expression sizes for automata obtained by our construction. This is done by constructing expressions that maximize this ratio. We will arrive at an infinite family of expressions that is structurally restricted up to repetition of a smallest such expression. We will further show that the automata produced by our constrution are minimal, i.e., that the derived bound is tight.

5.1 Worst Case Expressions

Definition 8. Let r be an expression. The *conversion ratio* of r, denoted $\mathrm{c}(r)$, is defined as

$$\mathrm{c}(r) := \frac{|A_r|}{|r|}.$$

An expression r is *worse* than an expression s, if $\mathrm{c}(r) > \mathrm{c}(s)$. If no expression is worse than r, then r is a *worst case* expression.

A worst case expression maximizes the size an FA can attain relative to the size of the expression it was constructed from. If μ is worst case (given that such

	$\Rightarrow_\varnothing$	$\Rightarrow_\cdot$	$\Rightarrow_+$	$\Rightarrow_*$	$\Rightarrow_X$	$\Rightarrow_\triangleleft$	$\Rightarrow_Y$	$\Rightarrow_\circlearrowleft$
$\Delta_\mathrm{i}(\lvert Q\rvert)$	0	1	0	1	-1	-1	-1	$-(n-1)$
$\Delta_\mathrm{i}(\lvert \delta\rvert)$	-1	1	1	2	0	-1	-1	$-n$

Table 5.1: Changes in state- and transition-complexity upon conversion in $\mathfrak{C}$. In the last column, n denotes the number of transition of an ε-cycle.

expressions exist in the first place) and r is arbitrary, the size of the output of our construction is bounded relative to that of its input as

$$|A_r| \leq \mathrm{c}(\mu)|r|.$$

Given a rewriting $A_r^0 \Rightarrow^\star A_r$, we obtain the size of A_r by counting the elements contributed to A_r by each rewriting step. These values are given in Tab. 5.1 for each rule: therein, $\Delta_\mathrm{c}(|Q|)$ denotes the number of states introduced or removed upon $A_r^k \Rightarrow_\mathrm{c} A_r^{k+1}$, and $\Delta_\mathrm{c}(|\delta|)$ denotes the change in the number of transitions. Notice that these changes are constant for every rule except cycle elimination. We will shortly see that this rule does not pose a problem for analyzing sizes.

Let $|r|_\mathrm{i}$ denote the number of times each rule of $\mathfrak{C}$ is applied in a fixed construction of A_r by rewriting A_r^0, for $\mathrm{i} \in \{\cdot, +, *, \circlearrowleft, \ldots\}$. The size of A_r is given by a weighted sum, as follows:

$$|A_r| = |A_r^0| \; + \sum_{\mathrm{i}\in\{\cdot,+,*,\circlearrowleft,\ldots\}} |r|_\mathrm{i} \left(\Delta_\mathrm{i}(|Q|) + \Delta_\mathrm{i}(|\delta|)\right).$$

Obviously, the frequency of expansions: $\Rightarrow_\varnothing$, $\Rightarrow_\cdot$, $\Rightarrow_+$, and $\Rightarrow_*$, in the construction of A_r, equals $|r|_\varnothing$, $|r|_\cdot$, $|r|_+$, and $|r|_*$, respectively. On the other hand, the frequency of eliminations depends on the chosen rewriting. For example, two transitions of an ε-cycle might be removed by fan elimination, followed by the elimination of a smaller cycle (cf. Fig. 4.5b). However, since A_r is unique, the weighted sum corresponding to any exhaustive conversion of some fixed A_r^k is constant.

To get an idea about the expected magnitude of the conversion ratio, we provide a first upper bound. This bound is determined using only the fact that the SSNF of an input is implicitly computed by our conversion. Elimination steps — the parts in our constructions that decrease that size of (intermediate) automata — are neglected.

Lemma 5.1.1. *For any expression r we find*

$$\mathrm{c}(r) < \frac{5}{3} + \frac{8}{3|r|}.$$

Proof. We start from $|A_r^0| = 3$ and take only expansions into account, i.e., rewritings that increase the size of intermediate EFAs. This yields

$$\begin{aligned} |A_r| &\leq 2|r|. + |r|_+ + 3|r|_* + 3 \\ &= |r| - |r|_+ + 2|r|_* + 2. \end{aligned}$$

Since r and $r^\bullet$ are converted to the same FA, we may assume that r is in SSNF already. Following Thm. 3.2.9 we then have $|r|_\omega \leq |r|_\mathcal{A}$, which implies $|r|_* \leq |r|_\mathcal{A}$ and thus $|r|_* \leq \frac{1}{3}(|r| + 1)$. Plugging this bound on $|r|_*$ into the above right hand side yields

$$|A_r| \;=\; |r| + \frac{2}{3}(|r| + 1) + 2 \;=\; \frac{5}{3}|r| + \frac{8}{3}.$$

Dividing the left and right hand sides by $|r|$ yields the claim. □

If we want to bound conversion ratio independently of the converted expression, we need to use the maximum value reached be the right hand side in Lem. 5.1.1. Since $|r| \geq 1$ for any r, this gives

$$\mathrm{c}(r) < \frac{5}{3} + \frac{8}{3}, \text{ resp. } \mathrm{c}(r) < 4\frac{1}{3}.$$

As the second fractional term in Lem. 5.1.1 gets arbitrarily close to zero with growing expression size, it is better to assume that r is at least of a certain size. This excludes only a finite number of expressions, so we still get a statement about almost all expressions. A corollary to Lem. 5.1.1 in this spirit is

Corollary 5.1.2. *For almost every expression r we find*

$$\mathrm{c}(r) < 2.$$

Proof. Let $|r| \geq 8$ in Lem. 5.1.1. □

The bounds on conversion ratio given in Lem. 5.1.1 and Cor. 5.1.2 will be improved in this chapter to a tight value, i.e., one that is reached by certain expressions. These are the worst case expressions, whose conversion ratio bounds that of any expression by definition. We infer the structure of these expressions by increasingly restricting the properties they may expose. In particular, we show that the structure of such an expression is unique up to repetition of a worst-case "atom". In other words, we infer an infinite family of structurally similar expressions whose conversion ration bounds that of *any* expression from above. Moreover, it will be shown that no structurally different expression reaches this bound.

From Lem. 5.1.1 we derive a criterion which sometimes allows us to decide among two expressions which is worse, when this is not obvious at a glance. This situation arises when expressions that are not explicitly given differ in size and yield automata which differ in size, too.

Corollary 5.1.3. *Let r and s be expressions where $|r| \geq 3$, $|s| = |r| + k$, and $|A_s| \geq |A_r| + l$. Then s is worse than r if*

$$\frac{l}{k} \geq 2.6.$$

Proof. By definition, s is worse than r if $\mathrm{c}(r) < \mathrm{c}(s)$. Using the fractional terms for conversion ratio, this inequality reads as

$$\frac{|A_r|}{|r|} < \frac{|A_s|}{|s|}, \text{ or equivalently } \frac{|A_r|}{|r|} < \frac{|A_r| + l}{|r| + k},$$

where r, s, A_r, and A_s are as in the claim. Rearranging the fractions shows that the latter inequality holds iff $\mathrm{c}(r) < \frac{l}{k}$. Since we assumed $|r| \geq 3$, Lem. 5.1.1 yields

$$\mathrm{c}(r) \leq \frac{5}{3} + \frac{8}{9} = 2.\overline{5} < 2.6.$$

Therefore, if $\frac{l}{k}$ is at least 2.6, it exceeds the conversion ratio of r, and the claim follows. □

To prove that a structural property φ — realized by a particular subexpression s — is absent from a worst case expression, we often argue indirectly. To this end we take an expression with this property, say, $r = c[s]$, replace the fixed occurrence of s with s', where s' does not satisfy φ. We then show that $r' = c[s']$ is worse than r.

Recall from Prop. 4.1.3 that replacing a unique subexpressions s with s' as above implies rewritings

$$A_r^0 \Rightarrow_{\mathfrak{B}}^k A_r^k \text{ and } A_r^0 \Rightarrow_{\mathfrak{B}}^k A_{r'}^k,$$

s.t. A_r^k and $A_{r'}^k$ differ exactly by the label of one transition, which is s in the former EFA and s' in the latter. Due to these labels, the two EFAs may admit different follow-up rewritings. Informally, these differences are restricted to a local level, so any difference in size among the EFAs obtained by these further rewritings becomes evident from local inspection, too. We can thus apply Cor. 5.1.3 (and similar results which are yet to be shown) to compare r and r' based on local replacements, i.e., without knowing the full expressions. A first opportunity for this kind of argumentation is the following lemma.

Lemma 5.1.4. *If r is worst case, then $|r|_\varnothing = 0$.*

Proof. Suppose that r is worst case and that $r = c[\varnothing]$ for some context c. Replace this occurrence of $\varnothing$ with any letter $a \in \mathcal{A}$ to get $r' := c[a]$. Now let E and E' be EFAs satisfying

$$A_r^0 \Rightarrow^n E \text{ and } A_{r'}^0 \Rightarrow^n E',$$

where E' can be derived from E by replacing the transition $(p, \varnothing, q)$ with (p, a, q). While the $\varnothing$-transition of E is removed at some future point in the conversion to A_r, its counterpart remains in E' resp. $A_{r'}$. Other than that, the number of operators is identical in r and r', so the number of expansions is identical in the construction of A_r and A_r.

We further find that no elimination step in the construction of A_r depends on the presence of the transition $(p, \varnothing, q)$, as opposed to (p, a, q). However, an elimination might well depend on the *absence* of a transition from p to q, so, if anything, the number of eliminations might be reduced by replacing $\varnothing$ with a.

Therefore, while $|r'| = |r|$, the FA $A_{r'}$ contains at least one element more than A_r. It follows that $\mathrm{c}(r) < \mathrm{c}(r')$, i.e., r is not worst case. Consequently, a worst case expression does not contain $\varnothing$, as claimed. □

Lemma 5.1.5. *If r is worst case with $|r| \geq 3$, then $|r|_\varepsilon = 0$.*

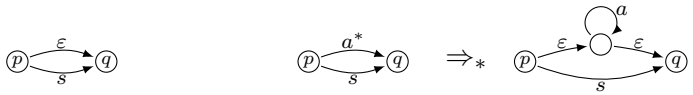
Proof. Suppose that r is worst case, $|r| \geq 3$, and that r contains ε. Fix an occurrence of ε within the context c, i.e., let $r = c[\varepsilon]$. Further let $t = (p, \varepsilon, q)$ denote any ε-transition with this fixed ε as its label in the construction of A_r. We distinguish whether any elimination depends on t in the construction of A_r, i.e., that the particular elimination would not be possibly without t.

- Assume that some elimination $A_r^k \Rightarrow_\mathrm{i} A_r^{n+1}$ does depend on t. Let $r' := c[a]$ for some $a \in \mathcal{A}$ and let $A_{r'}^k$ be the FA that is as A_r^k except that t is relabeled with a. Then the corresponding i-elimination is "blocked" in $A_{r'}^k$. Conversely, every elimination that is possible in $A_{r'}^k$ or a subsequent $A_{r'}^{k+d}$ has an analogue in A_r^k or A_r^{k+d-1}, respectively. The number of expansions is identical for r and r'. Thus follows $|A_r| \leq |A_{r'}| - 1$. Since $|r| = |r'|$, we find $\mathrm{c}(r) < \mathrm{c}(r')$, which contradicts the assumption that r is worst case.

- If no elimination ever depends on t in the construction of A_r, we consider the parent of our fixed ε in r. Observe that since $|r| \geq 3$, we have $r \neq \varepsilon$, i.e., the parent of each ε is defined. First off, since r is supposedly worst case, it is also in SSNF, hence ε is not the operand of an iteration.

  So assume that ε is a factor, wlog. in the product $s\varepsilon$. Any construction of A_r comes invariably to the point where the according $s\varepsilon$-transition is expanded, yielding a fan. This conflicts with the initial assumption of this case.

  Finally let ε be an addend, say, in $s+\varepsilon$. Here, we set $r' := c[a^*]$ for some $a \in \mathcal{A}$, and compare the constructions of A_r and $A_{r'}$. Let A_r^k and $A_{r'}^k$ denote k-step rewritings from A_r resp. $A_{r'}$ that are identical except that the former EFA contains an ε-transition from p to q, where the latter contains an a^*-transition. As sketched below, we locally compare A_r^k (left) to the EFA derived from $A_{r'}^k$ by expanding this transition (right).



  It becomes clear from the sketch that since t is no part of an elimination anchor in A_r^k or any derived EFA, neither are the "new" ε-transitions in $A_{r'}^{k+1}$ or any derived EFA. Moreover, as A_r^k and $A_{r'}^{k+1}$ are identical up to the discussed part, the rewriting of either EFA allows for an analogous one in the other EFA. We ultimately find $|A_{r'}| = |A_r| + 3$, while we have $|r'| = |r| + 1$. Since we assumed $|r| \geq 3$ we can apply Cor. 5.1.3 to find that r' is worse than r, contrary to the initial assumption.

As either case leads to a contradiction, the assumption $|r|_\varepsilon > 0$ does not hold for worst case r, as claimed. □

Recall that we have shown in Prop. 3.2.1 that the SSNF of an expression is not bigger than the expression itself. If an expressions is in SSNF already, the sizes are obviously equal. For reduced expressions, we find a stronger property.

Proposition 5.1.6. *Let r be a reduced compound expression. Then $r = r^\bullet$ iff $|r| = |r^\bullet|$.*

Proof. The implication from left to right is trivial. To prove the converse direction, assume that the r is a reduced compound expression and that $r \neq r^\bullet$. Then Thm. 3.2.8 implies $s^* \in \mathrm{sub}(r)$ with $\varepsilon \in \mathrm{L}(s)$ for some s. Since r is reduced, we also have that $\varepsilon \notin \mathrm{sub}(s)$, and $s \neq \varepsilon$ in particular, since s is a proper subexpression of r. So the fact that s is nullable is due to some subexpressions t^* or $t^?$ in s. For the SSNF of s^* we have $s^{*\bullet} = s^{\circ\bullet *}$, where the inner operator $^\circ$ traverses down s (cf. Def. 2) recursively, eventually "reaches" t^*, resp. $t^?$, and replaces either subexpression with t.

That is, at least one operator is removed upon computing $r^\bullet$. This decrease in size is not compensated by a part of the computation in another subexpression since $|s'^\bullet| \leq |s'|$ for any $s' \in \mathrm{sub}(r)$. This concludes the proof. □

This allows us to restrict the structure of a worst case expression significantly.

Theorem 5.1.7. *A worst case expression is reduced and in* SSNF.

Proof. Let μ be worst case, then μ is reduced, following Lems. 5.1.4 and 5.1.5. Suppose that μ is not in SSNF, then Prop. 5.1.6 yields $|\mu^\bullet| < |\mu|$, since μ is reduced. Since Cor. 4.3.3 assures that $A_{\mu^\bullet} = A_\mu$, we find

$$\mathrm{c}(\mu^\bullet) = \frac{|A_{\mu^\bullet}|}{|\mu^\bullet|} \geq \frac{|A_\mu|}{|\mu|} = \mathrm{c}(\mu).$$

This contradicts the assumption that μ is worst case, so μ must be in SSNF. □

Corollary 5.1.8. *If r is worst case, then $|r|_\circlearrowleft = 0$.*

Proof. The claim follows from Thm. 5.1.7 and Cor. 4.4.3 immediately. □

In the following, our arguments heavily depend on the fact that a worst case expression is in SSNF. According to Thm. 5.1.7, any candidate r for a worst case expression can be assumed to be in SSNF. If we find that r is not worst case by constructing a worse expression r', we need to take care to ensure that r' is still in SSNF.

Knowing that worst case expressions are in SSNF, we start by investigating stars, i.e., iterations, in such expressions. The characterization we found for

SSNF in Thm. 3.2.8 implies structural restrictions of worst case expressions wrt. iterations. More precisely, we find that each iteration in a worst case expression is an addend.

Lemma 5.1.9. *If μ is worst case and s^* is a proper subexpression of μ, then* $\mathrm{p}_\mu(s^*) = +$.

Proof. Let μ be worst case and $s^* \in \mathrm{sub}_{\neq}(\mu)$. We show that s^* is neither a base nor a factor. The first claim follows immediately: since μ is in SSNF according to Thm. 5.1.7, and s^* is certainly nullable, the fact that $\mathrm{p}_\mu(s^*) \neq *$ is implied by Thm. 3.2.8.

Suppose for the sake of contradiction that μ contains iterations that are factors. Among these iterations we choose a fixed "topmost" instance s^*. That is to say that while s^* is a factor, every iteration that properly contains the one we chose, i.e., every iteration that is a superexpression of s^* in μ, is an addend.

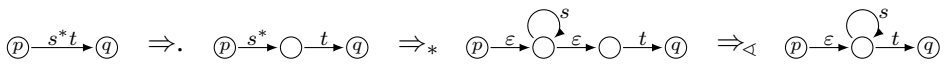
Formally, let $\mathrm{p}_\mu(s^*) = \cdot$ and assume that $s^* \in \mathrm{sub}(\sigma^*)$ and $s^* \neq \sigma^*$ implies $\mathrm{p}_s(\sigma^*) = +$. Further assume that s^* occurs in the product s^*t; the case ts^* is symmetric. We replace this product with a sum by setting $\mu' := \mu[{}^{s^*+t}/_{s^*t}]$. Observe that $|\mu| = |\mu'|$. Again, we let A^k_μ and $A^k_{\mu'}$ denote intermediate EFAs that differ only in that A^k_μ has a transition (p, s^*t, q) whereas $A^k_{\mu'}$ contains a transition $(p, s^* + t, q)$.

In the following, we distinguish whether μ' is in SSNF.

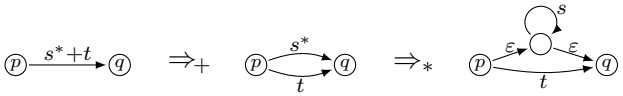
- If μ' is in SSNF, we consider EFAs obtained from rewriting A^k_μ and $A^k_{\mu'}$. In each case the rewriting starts in the "characteristic" transition which distinguishes A^k_μ and $A^k_{\mu'}$. Compare the number of elements from p to q in some particular A^{k+3}_μ, shown in Fig. 5.1a, to that which connect p to q in some $A^{k+2}_{\mu'}$, shown in Fig. 5.1b. We find an size increase of $A^{k+2}_{\mu'}$ over A^{k+3}_μ by one transition.

  If neither p nor q is removed in the remaining conversion, this difference in size carries over to A_μ and $A_{\mu'}$. So assume that one of p or q is removed in the construction of $A_{\mu'}$ from A^{k+2}_μ. Since this case assumes that μ' is in SSNF, we know that the removal does not result from cycle elimination. Other than that, cases for p and q are symmetric, so we consider removal of p only.

  First assume that at some point p becomes the center of a fan which is eliminated, i.e., $A^{k+i+2}_{\mu'} \Rightarrow_{\lhd[p]} A^{k+i+3}_{\mu'}$. Since r is assumed to be worst case, μ is $\varnothing$-free by Lem. 5.1.4 — thus μ' is $\varnothing$-free, too. So the outdegree of p, which is two in $A^{k+2}_{\mu'}$, does not decrease with subsequent conversions that



(a) Elements due to a starred factor



(b) Elements due to a starred addend

Figure 5.1: First case in the proof of Lem. 5.1.9, where we assume that either expression is in SSNF.

leave the state p intact. Consequently, p being the center of a fan in some $A_{\mu'}^{k+i+2}$ implies that p in entered by a single ε-transition in that EFA. We thus find an analogue EFA A_{μ}^{k+3+i} where the same elimination is possible.

If p is removed by X-elimination in some subsequent EFA $A_{\mu'}^{k+2+i}$, two ε-transitions are the only transitions that leave p in this EFA. One of these transitions results from expanding the transition labeled t, which is not incident to p in any A_{μ}^{k+3+i}. Since the in-transitions of p evolve the same way in either rewriting, this leads to a fan elimination with center p in some A_{μ}^{k+3+i}. With this the size of A_{μ} increases even more over that of $A_{\mu'}$.

Finally assume that p is removed by means of Y-elimination, in some A_{μ}^{k+2+i}. We then find again that at some point p is entered by a single, arbitrarily labeled transition, which also happens in some $A_{\mu'}^{k+3k+i}$. This allows for fan elimination in this EFA for μ, so the size increase for $A_{\mu'}$ over A_{μ} is maintained.

On the other hand, the removal of p in some A_{μ}^{k+3+i} does not necessarily have an analogue in some $A_{\mu'}^{k+2+i}$. If it does, however, the number of elements removed in the construction of $A_{\mu'}$ is equal to or less than the respective number removed from A_{μ}^{k+3+i}.

In any case, we find $|A_{\mu}| < |A_{\mu'}|$, which, since $|\mu| = |\mu'|$, implies that μ' is worse than μ.

- If μ' is not in SSNF, then Thm. 3.2.8 states that μ' contains a nullable base. This is a direct result of the change we made to μ, and the base that "violates" SSNF is the smallest base containing $s^* + t^*$ in μ'. Denote this base u' and let u denote the corresponding base in μ, i.e., let $u' = u[{}^{s^*+t}/_{s^*t}]$; now we can express μ' also as $\mu' = \mu[{}^{u'}/_{u}]$. Observe that our choice of s^* implies either

$u^* = \mu$ or that u^* is an addend in μ. The respective property holds for u' in μ'.

Moreover we find that t is not nullable: otherwise, s^*t would be nullable in u, just as $s^* + t$ is nullable in u'. Since u and u' differ only in that subexpression, the fact that u' is nullable is a result of our replacement, which allows to "propagate" the nullability of s^* upwards.

To reinstate SSNF, we modify the expression a second time, by "shifting" the star operating on u' in μ' from there to t. Formally, we set

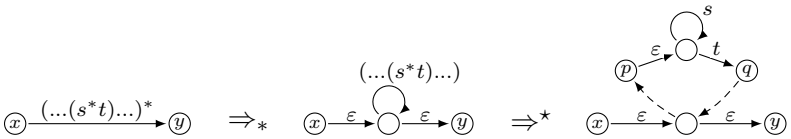
$$u'' := u'[{}^{s^*+t^*}/_{s^*+t}] \quad \text{and} \quad \mu'' := \mu'[{}^{u''}/_{u'^*}].$$

Since t is non-nullable and — being a subexpression of μ — is also in SSNF, we find that t^* is in SSNF, too. Since u' is the smallest base containing $s^* + t$ in μ', and u'^* is certainly nullable, the nullability of u'' does not lead to further nullable bases in μ''. No other base, resp. its nullability,is influenced by going from μ via μ' to μ'', so so according to Thm. 3.2.8 we have that μ'' is in SSNF. Observe that $|\mu''| = |\mu|$ holds, as we merely exchanged one instance of $\cdot$ for $+$ and repositioned a star.

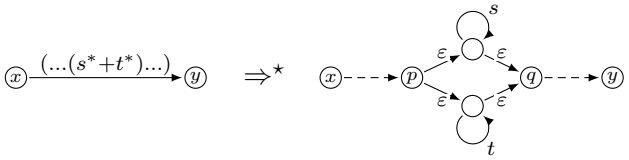
To show that $A_{\mu''}$ is bigger than A_μ let A^k_μ and $A^k_{\mu''}$ denote EFAs where $A^k_{\mu''}$ can be derived from A^k_μ by replacing a transition (x, u^*, y) with (x, u'', y), where, to stress the relevant differences,

$$u^* = (\ldots(s^*t)\ldots)^* \quad \text{and} \quad u'' = (\ldots(s^* + t^*)\ldots).$$

We compare two EFAs A^{k+i}_μ and $A^{k+j}_{\mu''}$, constructed from A^k_μ and $A^k_{\mu''}$ through conversions of elements that emerge from the transition that tells A^k_μ from $A^k_{\mu''}$. Either rewriting first fully expands the context of s^*t and $s^* + t^*$, and then further applies expansions to EFAs with a s- and a t-transition each, as shown in Fig. 5.2. We find that $A^{k+j}_{\mu''}$ contains one transition more than A^{k+i}_μ. Locally, the s- and t-transitions in either EFA allow for the same sequences of conversions, and every elimination that removes one of p, q, x, or y in $A^{k+j}_{\mu''}$ has an equivalent in A^{k+i}_μ that removes at least the same number of elements. It is crucial to note that, following our choice of s^* in μ, either x is the initial and y the final state in each of the considered EFAs, or there is an additional transition from x to y in each EFA. This reflects the cases $u^* = \mu$ and $\mathrm{p}_\mu(u^*) = +$. These properties prevent the eventuality of a fan



(a) Converting A_μ^k to A_μ^{k+i}



(b) Converting $A_{\mu''}^k$ to $A_{\mu''}^{k+j}$

Figure 5.2: Second case in the proof of Lem. 5.1.9. Notice that either is x the initial and y the final state in each depicted EFAs, or there is a further transition from x to y, which is not shown.

elimination with center x or y in A_μ^{k+i}, s.t. no counterpart is possible in $A_{\mu''}^{k+j}$ due to the lack of an appropriate ε-transition.

In either case, the assumption that a worst-case μ contains a topmost starred factor, is falsified. It follows that μ contains no starred factor at all. Therefore every iteration in a worst case expression is an addend, as stated. □

The preceding lemma implies that in a worst-case expression, the number of stars is at most twice the number of sum-operators. This allows us to improve the bound on conversion ratio which is given in Lem. 5.1.1, essentially to the value found by Ilie & Yu for a different construction [30].

Lemma 5.1.10. *For any expression r we find*

$$\mathrm{c}(r) < \frac{3}{2} + \frac{5}{2|r|}.$$

Proof. Let μ be worst-case and proceed as in the proof of Lem. 5.1.1, where we arrived at

$$|A_\mu| \leq |\mu| - |\mu|_+ + 2|\mu|_* + 2 \quad \text{and} \quad |\mu|_* \leq \frac{1}{3}(|\mu| + 1)$$

As we just mentioned, Lem. 5.1.9 implies $|\mu|_+ \geq \frac{1}{2}|\mu|_*$. Plugging this into above inequations yields

$$|A_\mu| \quad \leq \quad |\mu| + \frac{3}{2}|\mu|_* + 2 \quad \leq \quad \frac{3}{2}|\mu| + \frac{5}{2}.$$

We divide the left- and right-hand sides by $|\mu|$. This yields an upper bound for the conversion ratio of μ. Since μ is worst-case by assumption, its conversion ratio is an upper bound to that of any expression. □

As before, this upper bound allows us to compare certain expressions that differ in size and are converted to automata that differ in size, too.

Corollary 5.1.11. *Let r and s be expressions s.t. $|r| \geq 16$, $|s| = |r| + k$ and $|A_s| = |A_r| + l$. Then s is worse than r if*

$$\frac{l}{k} \geq \frac{5}{3}.$$

Proof. The proof is the same as for Cor. 5.1.3, except that we use $|r| \geq 16$ and apply Lem. 5.1.10. This yields

$$\mathrm{c}(r) \leq \frac{3}{2} + \frac{5}{32} = 1.65625.$$

Since $\frac{3}{2}$ exceeds this value, the claim follows. □

This stronger version of Cor. 5.1.3 is necessary to show that the remaining eliminations — fan, X- and Y-elimination — are also absent from the conversion of a worst case expression. We can apply the same kind of proof with smaller increments in FA size relative to expression size. In particular, this criterion applies if an increment of expression size by three increases the size of the corresponding FA by at least five.

An inconvenience of Cor. 5.1.11 is its restriction to expressions of size at least 16. All statements building upon the corollary must include a clause similar to "... and suppose further that $|\mu| \geq 16$...". Since this excludes only a finite number of expressions, and we seek to infer an infinite family of worst-case expressions, the shortcoming is not severe and will be ignored in the remainder of the analysis.

Lemma 5.1.12. *If μ is worst-case, then $|\mu|_\lhd = 0$.*

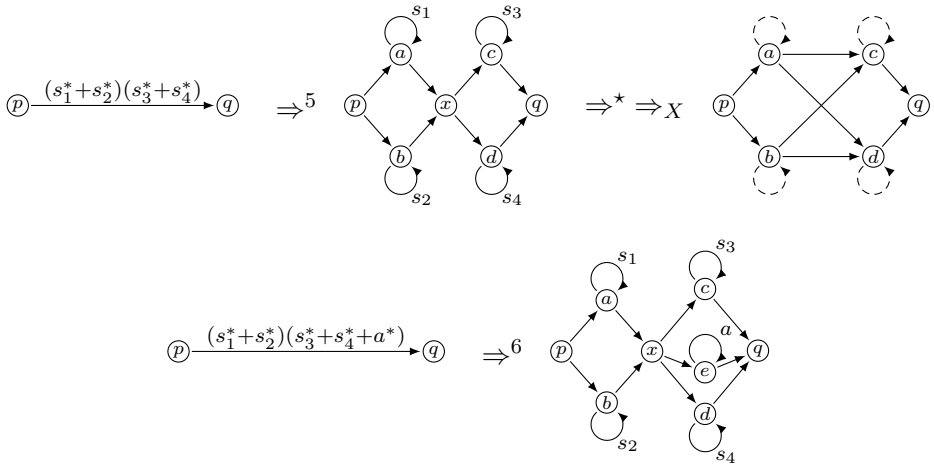


Figure 5.3: Proof of Lem. 5.1.13. Augmenting the sum $s_3^* + s_4^*$ by a further addend a^* yields four new elements and prevents X-elimination of the state x.

Proof. If μ is worst-case then $|\mu|_\varepsilon = 0$ by Lem. 5.1.5, so the ε-transition appearing in a fan must be the result of a preceding star expansion. If a fan emerges from $\Rightarrow_*$, a starred factor is required. Following Lem. 5.1.9, these do not occur in a worst-case expression. □

Lemma 5.1.13. *If μ is worst-case, then $|\mu|_X = 0$.*

Proof. As noted in the proof of Lem. 5.1.12, an ε-transition in any A_μ^k results from star expansion. In particular, an X-anchor can only result from expanding a subexpression with structure $\chi = (s_1^* + s_2^*)(s_3^* + s_4^*)$. So suppose $\chi \in \mathrm{sub}(r)$, let $\chi' = (s_1^* + s_2^*)(s_3^* + s_4^* + a^*)$ for some $a \in \mathcal{A}$, and set

$$\mu' := \mu[\chi'/\chi].$$

For the relative sizes of expressions and automata, we find $|\mu'| = |\mu| + 3$, and $|A_{\mu'}| = |A_\mu| + 5$. The increase in automaton size is due to two additional expansions and blocking the X-elimination (cf. Fig. 5.3). Applying Cor. 5.1.11 yields that μ' is worse than μ. □

Lemma 5.1.14. *If μ is worst-case, then $|\mu|_Y = 0$.*

Proof. Let μ be worst case, and observe that Lem. 5.1.13 implies that $A_\mu^{\mathfrak{B}}$ is $\mathfrak{X}$-normal. Suppose that $A_\mu^{\mathfrak{B}}$ contains a $Y[q]$-anchor. We assume wlog. that the out-transitions of q are all ε-transitions and the sole in-transition is $t = (p, a, q)$. We replace a uniquely determined occurrence of a in μ, the one which becomes the label of t in $A_\mu^{\mathfrak{B}}$, by setting $\nu := \mu[^{a+b^*}/_a]$ for $b \in \mathcal{A}$.

Then $|A_\nu| = |A_\mu|+6$, since "disabling" Y-elimination leaves q and the transition (p, a, q) untouched, while four additional elements are introduced. For relative sizes we have $|\nu| = |\mu| + 3$, so we apply Cor. 5.1.11 to conclude that ν is worse than μ. This contradicts our assumption that μ is worst case, so $A_\mu^{\mathfrak{B}}$ is $\mathfrak{Y}$-normal, which implies the claim. □

We have shown that in the construction of A_r for any worst case r, no elimination occurs at all. With this, the inequality derived in the proof of Lem. 5.1.10 becomes an equality.

Corollary 5.1.15. *If μ is worst-case, then*

$$|A_\mu| = |\mu| + 2|\mu|_* - |\mu|_+ + 2.$$

This shows that fixing the size of a worst-case expression leaves the number of sums and stars as the sole parameters that determine the size of the resulting FA. We proceed by investigating the interrelations between those two operators. Our findings will narrow the structural properties of worst case expressions further down, to the effect that a unique structure emerges.

Lemma 5.1.16. *In a worst case expression, every addend is nullable.*

Proof. Let μ be worst-case and suppose that μ contains an addend that is not nullable, say, $s_1 + s_2 \in \mathrm{sub}(\mu)$ where $\varepsilon \notin \mathrm{L}(s_1)$. We replace s_1 with s_1^* and introduce a further nullable addend into the sum we consider. To this end, let a be a letter and set $\mu' := \mu[^{s_1^*+a^*}/_{s_1}]$. Now distinguish whether μ' is in SSNF.

1. If μ' is in SSNF, then $|A_{\mu'}| = |A_\mu| + 7$, while $|\mu'| = |\mu| + 4$. The difference in the sizes of intermediate EFAs is clear from Fig. 5.4. Based on the in- and out-degrees of states incident to the transition that is replaced according to the replacement that yields μ' from μ, we also find that no eliminations are possible in the construction of $A_{\mu'}$ that have no analogue in the construction of A_μ. In particular, the extra addend a^* prevents that the new ε-transitions, which result from replacing s_1 with s_1^*, are removed by a subsequent X-elimination.

   It follows from Cor. 5.1.11 that μ' is worse than μ.

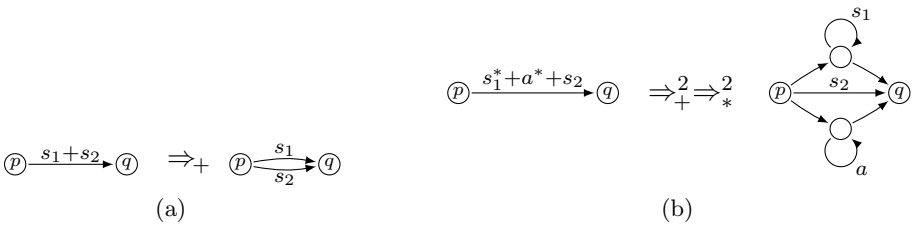


Figure 5.4: First case in the proof of Lem. 5.1.16.

2. If μ' is not in SSNF, replacing s_1 with $s_1^* + a^*$ introduced a nullable base. Since s_1 is not nullable by assumption and a is a letter, the base in question is the smallest base that contains $s_1^* + a^*$ in μ'. Let u' denote this base and let u be the corresponding (non-nullable) base in μ, so we have $u' = u[{}^{s_1^*+a^*+s_2}/_{s_1+s_2}]$. The nullability of every base besides u is the same in μ and μ'. We remove the star operating on u' in μ' by setting $\mu'' = \mu'[{}^{u'}/_{u'^*}]$. With this, the property that no base is nullable, is reinstated and μ'' is in SSNF by Thm. 3.2.8.

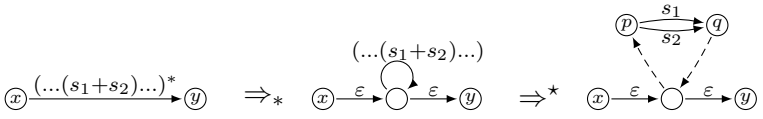
   Our standard argument — an increase in size for intermediate EFAs which is maintained throughout the remaining conversions — applies to the construction of A_μ and $A_{\mu''}$. In this case, we find $|A_{\mu''}| = |A_\mu| + 5$ for automaton size (cf. Fig. 5.5); since the sizes of expressions relate to another as $|\mu''| = |\mu| + 3$, Cor. 5.1.11 yields that μ'' is worse than μ.

In either case, we have construced an expression that is worse than μ. Therefore, the assumption that a worst case expression contains non-nullable addend is false, which is equivalent to the claim. □

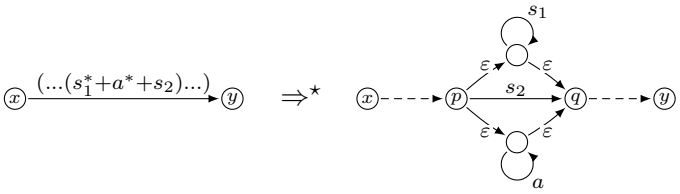
Having found that each addend is nullable in a worst case expression, we ask for the full structure of an addend. Recall know from Lem. 5.1.5 that a worst case expression is ε-free. Therefore, the fact that an addend is nullable implies that it contains at least one iteration as a subexpression; this is the only way to denote ε in a worst case expression.

To proceed, we introduce generalized notions of sums and products in an expression. An expression of the form $s = s_1 + s_2 + \cdots + s_k$ is called a sum of arity k and written concisely as

$$s = \sum_{1 \le i \le k} s_i.$$



(a)



(b)

Figure 5.5: Second case in the proof of Lem. 5.1.16.

Likewise, an expression as $s = s_1 s_2 \cdots s_k$ is called a product of arity k and written

$$s = \prod_{1 \leq i \leq k} s_i.$$

We introduce some further notions for generalized sums; these notions carry over to generalized products by applying the obvious changes. A generalized sum s that is a subexpression of r is called *maximal* in r, if s is not an addend itself and no s_i is a sum. We call s *star-maximal* if s is maximal and each s_i is an iteration. Clearly, every sum in an expression is either maximal or part of a maximal sum; it is at times more convenient to argue with maximal sums. For a maximal sum in a worst case expression, we get the following result.

Lemma 5.1.17. *In a worst case expression, each maximal sum is star-maximal.*

Proof. Let μ be worst-case and suppose μ contains maximal sums that are not star-maximal. We choose a smallest sum of that kind: let $\sigma = \sum \sigma_i$ be maximal but not star-maximal and assume that all maximal sums that are proper subexpressions of σ are star-maximal. Furthermore, let σ_k be an addend which is not an iteration. Since σ is maximal, σ_k is not a sum. Since σ_k is nullable by Lem. 5.1.16, it is not a letter either. Therefore σ_k must be a product, which we suppose to be maximal. Commutativity of sums allows us to assume $k = 1$. We are thus looking at

$$\sigma = \sigma_1 + \sum_{i=2}^{n} \sigma_i = \prod_{i=1}^{m} \pi_i + \sum_{i=2}^{n} \sigma_i.$$

Since σ_1 is nullable, every factor π_i of σ_1 is nullable, too. Moreover, $|\mu|_\varepsilon = 0$ implies that this is due to iterations that occur in each π_i. Our choice of σ and the fact that each iteration in μ is an addend (Lem. 5.1.9) imply that each π_i contains sums which are all star-maximal. Since σ_1 is assumed to be a maximal product, each π_i is necessarily such a sum. In greater detail, σ admits the structure

$$\sigma = \prod_{i=1}^{m} \sum_{j=1}^{l_i} \varsigma_{ij}^* + \sum_{i=2}^{n} \sigma_i.$$

Depending on the number n of addends in σ we construct an expression which is worse than μ. Notice that σ is of arity at least two, i.e., we have $n \geq 2$.

1. $n = 2$: Since $\sigma = \sigma_1 + \sigma_2$ is maximal, σ_2 is not a sum itself, and since σ_2 is nullable (Lem. 5.1.16) it is not a literal. This leaves two possibilities:

   a) If σ_2 is an iteration, $\sigma_2 = \kappa^*$, we construct σ' from σ as

   $$\sigma' := (x \cdot \sigma_1)^* + \sigma_2 = (x(\prod \sum \varsigma_{ij}^*))^* + \kappa^*,$$

   and let $\mu' := \mu[\sigma'/\sigma]$. Every elimination in the construction of $A_{\mu'}$ has a counterpart in the construction of A_μ, so we only need to compare the difference in positive contributions to FA size: looking up Tab. 5.1 we find $|A_{\mu'}| = |A_\mu| + 5$. Since we have $|\mu'| = |\mu| + 3$, Cor. 5.1.11 applies to yield the statement.

   b) If σ_2 is a (maximal) product, again Lem. 5.1.16 implies that its factors are star-maximal sums, i.e.,

   $$\sigma_2 = \prod_{i=1}^{m'} \sum_{j=1}^{l'_i} \varsigma_{ij}^* \quad \text{, which we write as} \quad \sigma_2 = \prod_{i=m+1}^{m+m'} \sum_{j=1}^{l_i} \varsigma_{ij}^*.$$

   We construct the product σ' from σ by exchanging its main operator for a concatenation-symbol and introducing a new starred addend in the first factor of σ_2:

   $$\sigma' = (\prod_{i=1}^{m} \sum_{j=1}^{l_i} \varsigma_{ij}^*)(x^* + \sum_{j=1}^{l_{m+1}} \varsigma_{1j}^*)(\prod_{i=m+2}^{m+m'} \sum_{j=1}^{l_i} \varsigma_{ij}^*)$$

   With this, we set $\mu' := \mu[\sigma'/\sigma]$; exchanging the sum for a product introduces an additional state q. No new eliminations become possible

from this replacement; in particular, the extra addend x^* in σ' ensures $\mathrm{d}^+(q) \geq 3$, so no $X[q]$-anchor emerges. We find $|\mu'| = |\mu| + 3$ and $|A_{\mu'}| = |A_\mu| + 5$, so μ' is worse than μ according to Cor. 5.1.11.

2. $n \geq 3$: Again we exchange a sum for a product by setting

$$\sigma' = \sigma_1 \cdot (x^* + \sum_{i=2}^{n} \sigma_i).$$

As before, the product introduces a new state while the extra addend x^* prevents the eventuality of X-elimination. This yields $|A_{\mu'}| = |A_\mu| + 5$ and $|\mu'| = |\mu| + 3$, so Cor. 5.1.11 yields the claim.

□

Lemma 5.1.18. *In a worst case expression, every letter is a base.*

Proof. Let μ be worst-case and fix an occurrence of the letter a in μ. If a is not a base, it is either an addend or a factor. However, since all sums in μ are star-maximal according to Lem. 5.1.17, a is not an addend.

So suppose that a is a factor in a maximal product $\pi = \prod \pi_i$, i.e., $a = \pi_k$ for some appropriate k. Then there is at least one factor π_{k-1} or π_{k+1} "next" to a. Let wlog. π_{k+1} be this factor. Since we consider a maximal product π, the factor π_{k+1} is not a product itself. Moreover, since each iteration in μ is an addend, by Lem. 5.1.9, π_{k+1} is not an iteration either. Therefore, π_{k+1} must be a (maximal) sum or a letter. However, either possibility leads to a contradiction:

- If π_{k+1} is a sum, it is star-maximal, i.e., $\pi_{k+1} = \sum_i \varsigma_i^*$. The subexpression $x \sum_i \varsigma_i^*$ would lead to Y-anchors, thus contradicting Lem. 5.1.14.

- If π_{k+1} is a letter y, we find that $\nu := \mu[x+y^*/xy]$ is worse than μ. This follows from Cor. 5.1.11 due to $|\nu| = |\mu| + 1$ and $A_\nu = A_\mu + 2$.

□

At this point, the properties a worst case expression may exhibit are restricted to an extent where a characteristic structure emerges.

Lemma 5.1.19. *If r is worst case, then the structure of r is*

$$r = \prod_{i=1}^{n} \sum_{j=1}^{k_i} a_{ij}^*$$

for $n, k_j \geq 2$, and $a_{ij} \in \mathcal{A}$.

Proof. Let μ be worst case. As stated in Lem. 5.1.18, every letter occurs as a base in μ, thus $|\mu|_* \geq |\mu|_{\mathcal{A}}$. Following Thm. 5.1.7, μ is SSNF, so by virtue of Thm. 3.2.9, μ does not contain any further base, resp. iteration. Put differently, every base in μ is a letter, i.e., there is a one-to-one correspondence of letters and iterations in μ.

So μ contains iterations, each of which, following Lem. 5.1.9, is an addend. Consequently, μ contains sums, resp. maximal sums. Let the maximal sums of μ be enumerated in any fashion, and let σ_i denote the i-th maximal sum in that enumeration. According to Lem. 5.1.17, every σ_i is star-maximal; and as we discussed above, every addend of σ_i is simply a starred letter. Thus the structure of σ_i, for each i, is as follows:

$$\sigma_i = \sum_{j=1}^{k_i} a_{ij}^* \text{ for } a_{ij} \in \mathcal{A}.$$

Since the σ_i are maximal, no σ_i is a proper subexpression of some σ_j. If μ contains more than just one such sum, some of the σ_i occur as factors in a common product. Let π be such a product and assume wlog. that π is maximal. Consider the parent of π in μ. Since π is maximal, it is not a factor itself, and since all letters are bases already, π is not a base either. Consequently, π is not an addend, since every addend is an iteration in μ. It follows that no operator can be the parent of π in μ, which is only possible if $\mu = \pi$. This provides the basic structure

$$\mu = \pi = \prod_{i=1}^{n} \sum_{j=1}^{k_i} a_{ij}^*,$$

as claimed. □

Observe that the expression in Lem. 5.1.19, leaves the number of factors and the number of addends in each sum unspecified. It turns out that the number of addends needs to alternate between two and three to ensure that an expression with that structure is worst-case. The number of factors, on the other hand, may be arbitrary. This fully specifies the structure of a worst-case expression.

Theorem 5.1.20. *An expression r is worst-case iff*

$$r = \prod_{i=0}^{n} \sum_{j=1}^{2+i \bmod 2} a_{ij}^* \quad \text{or} \quad r = \prod_{i=1}^{n} \sum_{j=1}^{2+i \bmod 2} a_{ij}^*$$

for any $n \in \mathbb{N}$ and $a_{ij} \in \mathcal{A}$.

Proof. In this general case, and subject to the condition that no X-eliminations occur in the construction, we compare the sizes of μ and A_μ, which are

$$\begin{aligned} |\mu| &= (n-1) + \sum_{i=1}^{n} (3k_i - 1) = 3\sum_{i=1}^{n} k_i - 1, \text{ and} \\ |A_\mu| &= \sum_{i=1}^{n} 4k_i + n - 1 = 4\sum_{i=1}^{n} k_i + n - 1. \end{aligned}$$

This leads to the following conversion ratio:

$$\mathrm{c}(\mu) = \frac{|A_\mu|}{|\mu|} = \frac{4\sum k_i + n - 1}{3\sum k_i - 1} = 1 + \frac{\sum k_i + n}{3\sum k_i - 1}.$$

The term on the right shows that $\mathrm{c}(\mu)$ is maximal iff n is maximal relative to $\sum k_i$, or equivalently, iff $\sum k_i$ is minimal for fixed n. All sums in μ are proper, i.e., not unary, so $k_i \geq 2$ follows for all i. However, simply setting $k_i = 2$ for all i introduces an X-anchor in X_μ for every pair of adjacent factors. The problem disappears if k_i alternates between 2 an 3, i.e., for $k_i = 2 + i \bmod 2$ or $k_i = 2 + (i+1) \bmod 2$. The particular term determines whether this alternation starts with 2 or with 3. It is easily seen using the pigeonhole principle that any smaller $\sum k_i$ leads to X-anchors. □

The structure of a worst case expression is thus highly repetitive: a succession of sums that alternate between two and three starred letters as addends. Clearly, the choice of these letters does not influence the size of an automaton constructed from such an expression. The fact whether the expression starts with sum of two or of three addends is irrelevant for automaton size, too.

According to Cor. 5.1.15, the only parameter that controls the size of the FA constructed for such an expression is the overall number of repetitions, i.e., the number of factors in it. For ease of analysis we assume that sums of alternating length appear in pairs and start with a binary sum. Moreover, we fix an alphabet of size five, say, $\mathcal{A} := \{a_1, a_2, a_3, a_4, a_5\}$.

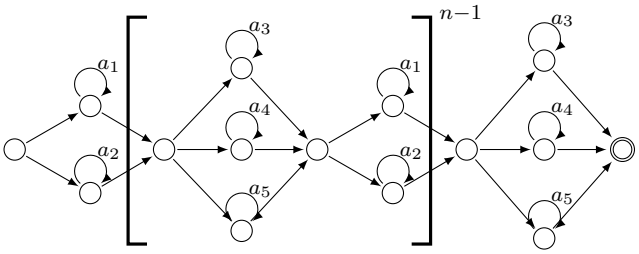


Figure 5.6: Automaton constructed from μ_n (ε-labels are omitted). The bracketed subautomaton is repeated according to the parameter n.

With above conventions we a define a parametrized "showcase" expression $\mu_n \in \mathrm{RE}_{\mathcal{A}}$ as

$$\mu_n := \prod_{i=1}^{2n} \sum_{j=1}^{2+i \bmod 2} a_{ij}^*,$$

which is equivalently conveyed by the less formal but better readable notation

$$\mu_n = [(a_1^* + a_1^*)(a_3^* + a_4^* + a_5^*)]^n.$$

The corresponding parametrized automaton A_{μ_n}, as derived by our construction, is shown in Fig. 5.6.

As a corollary, this allows us to give an upper bound for the size of an automaton constructed for *any* expression in the size of this expression.

Theorem 5.1.21. *Any expression r can be converted to a unique normalized FA A s.t.*

$$|A| < \frac{22}{15}|r| + 2.5, \text{ or equivalently } |A| < 1.4\overline{6}|r| + 2.5.$$

Proof. Given an expression r, choose n s.t. μ_n is the smallest worst case expression, as defined above, that is not smaller than r. For the size of μ_n and the FA constructed from it, we find $|\mu_n| = 15n - 1$ and $|A_{\mu_n}| = 22n + 1$. The conversion ratio of μ_n is therefore

$$\mathrm{c}(\mu_n) = \frac{22n+1}{15n-1} = \frac{22}{15} + \frac{37}{15|\mu_n|}.$$

By choice, we have $|r| \leq |\mu_n|$, and since μ_n is worst case, we also have $\mathrm{c}(r) \leq \mathrm{c}(\mu_n)$. For the size of A_r we thus find

$$\begin{aligned} |A_r| \;&=\; \mathrm{c}(r)|r| \;\leq\; \mathrm{c}(\mu_n)|r| \;=\; \frac{22}{15}|r| + \frac{37}{15|\mu_n|}|r| \\ &\leq\; \frac{22}{15}|r| + \frac{37}{15} \;=\; \frac{22}{15}|r| + 2\frac{7}{15}. \end{aligned}$$

As we have established in Sec. 4.2, the normalized FA A_r is unique. In conclusion, setting $A := A_r$ satisfies the claim. □

5.2 A Lower Bound on Conversion Ratio

In this section we show that our construction is worst-case optimal, which is to say that the automaton constructed for a worst case expression is of minimal size. Hence there is no construction that can attain a better conversion ratio in general.

As a preliminary, we prove a lower bound for the combined number of arcs and vertices necessary to assert certain paths among two sets of vertices in a graph.

Proposition 5.2.1. *Let L and R be disjoint sets of vertices in a graph G, and assume that the following conditions are satisfied*

1. *There is an (l, r)-path for arbitrary $l \in L$, $r \in R$ in G.*
2. *There is no (L, L)-path and no (R, R)-path in G.*
3. *There is no (R, L)-path in G.*

Then G contains at least $\min\{|L||R|, |L| + |R| + 1\}$ *additional elements.*

Proof. Let G be as in the claim and observe observe that the (L, R)-paths need not be disjoint. We distinguish whether a vertex lies on some (L, R)-path.

- If there is no such vertex on any (L, R)-path, each $l \in L$ is adjacent to every $r \in R$. Thus, G contains at least $|L||R|$ arcs.

- If x lies on the (l, r)-path P then $x \notin L \cup R$, due to the second precondition of the claim. Now P contains at least three elements, x and two arcs. Consider the remaining vertices of L and R: every $r' \in R \setminus r$ is the endpoint of a path from l, so every such r' is the endpoint of an arc $a_{r'}$. Likewise, every $l' \in L \setminus l$ is the tail of an arc $a_{l'}$. By above path-properties, no pair $a_{r'}$, $a_{l'}$ coincides. Hence G contains $3 + (|L| - 1) + (|R| - 1) = |L| + |R| + 1$ additional elements.

In either case, G contains at least $\min\{|L||R|, |L| + |R| + 1\}$ additional elements, as claimed. □

In the following, we refer to a maximal nonempty repetition of a letter in a word as a *block*. Formally, a block of w is any subword $v = a_i^k$, for $k \geq 1$, s.t. $w = uvy$, $u \neq u'a_i$, and $y \neq a_i y'$. A block a_i^k is also called an a_i-block. For example, the word $a_1a_1a_1a_2a_1a_3a_3$ contains four blocks: a_1^3, a_2, a_1, a_3^2; these are two a_1-blocks, an a_2-block, and an a_3-block. However, the word does not contain the block a_1^2.

We reconsider the parametrized worst case expression μ_n which we defined as

$$\mu_n = [(a_1^* + a_2^*)(a_3^* + a_4^* + a_5^*)]^n.$$

Observe that the number of blocks in any $w \in \mathrm{L}(\mu_n)$ is at most $2n$. We call a word that reaches that bound *block-maximal*. The structure of a block-maximal word w is

$$w = b_{1,j_1} b_{2,j_2} \ldots b_{2n,j_{2n}},$$

where $j_i \in \{1, 2\}$ if i is odd and $j_i \in \{3, 4, 5\}$ if i is even. Conversely, any word that satisfies these properties is a block maximal word of $\mathrm{L}(\mu_n)$. The restricted indexing of blocks in a block-maximal word is the key for proving the following lower bound.

Theorem 5.2.2. *A normalized FA that accepts* $\mathrm{L}(\mu_n)$ *is at least of size* $22n+1$.

Proof. Let A be any normalized FA accepting $\mathrm{L}(\mu_n)$, and consider how a block-maximal word $w_{\max}$ is accepted by A. According to the structure of μ_n, each block of $w_{\max}$ can be of arbitrary length and is therefore read in a cyclic substructure of A. Let $C_{i,j}$ denote the substructure of A that accepts a block $b_{i,j}$ or at least an arbitrarily long part of it.

The proof consists of two parts. First, we bound the number of cyclic structures $C_{i,j}$ in A from below. More precisely, we show that the number of such structures is bounded by the number of ways a block can be indexed in $w_{\max}$. In the

second part, the number of elements that are required to connect these cycles in A, without distorting the accepted language, is bounded from below.

1. Let $b_{i,j}$ and $b_{n,m}$ denote two distinct blocks in some $w_{\max}$, i.e., let $i \neq n$ or $j \neq m$. We distinguish two main cases that cover all possibilities for this.

   First assume $i \neq n$, where wlog. $i < n$. This means that $b_{i,j}$ may precede $b_{n,m}$ is some $w_{\max}$. For the sake of contradiction suppose that the associated cycles $C_{i,j}$ and $C_{n,m}$ of A share a state q.

   - If the parity of i and n differs, then $j \neq m$ follows according to above observation about the block-structure of $w_{\max}$. The number of blocks in $w \in L(A)$ is unbound then: any sequence of alternating x_j- and x_m-blocks is accepted by alternating between $C_{i,j}$ and $C_{n,m}$ in A, changing the accepting cycle by means of the common state q.
   - If i and n are of the same parity, yet $i < n$, there is an "intermediate" block $b_{k,l}$ in w, s.t. $i < k < n$ and the parity of k differs from that of i, resp. n. Therefore $x_j \neq x_k$ and $x_k \neq x_n$, even though $x_i = x_n$ is possible.

   Second, we consider two blocks that may both occur as the i-th block in $w_{\max}$, say, $b_{i,j}$ and $b_{i,m}$, where $j \neq m$. Then, as in either subcase of the previous case, a state shared by $C_{i,j}$ and $C_{i,m}$ allows for words with an unbound number of alternating x_j- and x_m-blocks.

   Either case contradicts the fact that the number of blocks in $w \in \mathrm{L}(\mu_n)$ is bounded from above. Therefore the structures $C_{i,j}$ and $C_{n,m}$ are disjoint for each distinct pair of indexings $b_{i,j}$ and $b_{n,m}$.

2. To bound the number of elements that connect the cycles $C_{i,j}$ in A, we first show that A contains a path from $C_{i,j}$ to a distinct $C_{n,m}$ iff $i < n$.

   - If $i < n$, there is a block-maximal $w \in \mathrm{L}(\mu_n)$ with blocks $b_{i,j}$ and $b_{n,m}$. To accept w, A must contain a path from $C_{i,j}$ to $C_{n,m}$.
   - If $i \geq n$, suppose for the sake of contradiction that A contains a path from a state p of $C_{i,j}$ to a state q of $C_{i,m}$. Let x denote the (possibly empty) word that is read by traversing this path. We treat the cases $i = n$ and $i > n$ separately.

     Let $i = n$ and refer to the common value by just i. Since A is normalized, there are block-maximal words $w, w' \in \mathrm{L}(\mu_n)$ s.t. the i-th block of w and w' is an a_j-block, resp. an a_m-block, which is read by A in $C_{i,j}$, resp. $C_{i,m}$.

These two words can be written as $w = uvy$ and $w' = u'v'y'$, where v and v' denote the a_j- and a_m-blocks we are keeping track of. Now a word

$$z = uva_j^k x a_m^l v'y'$$

is accepted by A for some $k, l \in \mathbb{N}$, as follows: First, the prefix uv is read as a prefix of w. Thus A is reaches some state of $C_{i,j}$, from where it reads a sequence a_j^k to reach the state p of $C_{i,j}$. Next, A reads the subword x on the path from p to q. Now A is in $C_{i,m}$, wherein it reads a_m^l until reaching the state of $C_{i,m}$ from where the suffix $u'y'$ of w' leads to the final state A. However, we find $z \notin \mathrm{L}(\mu_n)$, as z consists of at least $2n+1$ blocks. Therefore, no path from $C_{i,j}$ to $C_{i,m}$, resp. to $C_{n,m}$, exists in A for this case.

Now let $i > n$, then, as we know from the previous case, A contains a path from $C_{n,m}$ to $C_{i,j}$. In the presence of a path from $C_{i,j}$ to $C_{n,m}$, A accepts words with an unbounded number of $b_{n,m}$- and $b_{i,j}$-blocks, since either cycle is reached from the initial state and reaches the final state. This again contradicts the property that the number of blocks in words from $\mathrm{L}(\mu_n)$ is bounded.

We integrate these findings into an overall lower bound on $|A|$. As we have seen, there is a distinct cycle $C_{i,j}$ in A for each possible indexing of a block in a block-maximal word of $\mathrm{L}(\mu_n)$. For such a block $b_{i,j}$, following the structure of μ_n, we find $j \in \{1, 2\}$ if i is odd and $j \in \{1, 2, 3\}$ if i is even. Thus there are $5n$ possible indexings of a block in a block maximal word. Consequently, A contains at least $5n$ cycles. Since a cycle consists of at least a state and a transition (possibly a loop), there are at least $10n$ elements in A to realize these cycles.

Next we analyze the number of elements required to connect these cycles among each other. To this end we arrange the cycles according to the block structure of block-maximal words in $\mathrm{L}(\mu_n)$. For fixed $1 \leq i \leq 2n$ we refer to the $C_{i,j}$ as the i-th layer of A. The structure of μ_n is transferred to A in that A contains $2n$ layers alternating between two and three cycles. As we have shown, A contains a path from each cycle of layer i to every cycle of layer k iff $i < k$. Transitive reduction shows that this at least requires a path from every cycle of layer n to every cycle of layer $n+1$, the conditions that no path exists among cycles within the same layer or from the $n+1$-th to the n-th layer, remains. For a pair of adjacent layers we choose a state from each cycle in either layer which gives us two sets of states that satisfy the preconditions of Prop. 5.2.1. As these

sets of states always contain two and three elements, Prop. 5.2.1 yields that they are connected by at least 6 additional elements in A. This holds for every pair of consecutive layers, of which there are $2n - 1$ many, so A contains at least $12n - 6$ additional elements to connect the $5n$ cycles we found.

A final contribution to the size of A are the initial state q_0 and final state q_f, and the transitions connecting these states to the layers we just investigated. Obviously, q_0 is no part of any layer, as each layer contains at least two cycles that are not connected by paths, while each cycle can be reached from q_0. In particular, the first layer of A can be reached from q_0. Thus there is a path from q_0 to either cycle of the first layer; since neither such path is a segment of the other, there are at least two extra transitions for reaching the structures of the first layer of A from its initial state. A symmetric argument shows that q_f is a further state of A, and that three additional transitions are at least required to guarantee that q_f is reached from cycles in the $2n$-th layer of A. So there are at least 7 more elements in A.

In total, this yields that for any normalized FA A with $\mathrm{L}(A) = \mathrm{L}(\mu_n)$, we find

$$|A| \leq 10n + (12n - 6) + 7 = 22n + 1,$$

as claimed in the statement. □

Therefore, there is no algorithm that constructs automata from expressions with a better ratio of input to output size than the one we are considering. This allows us to proclaim

Corollary 5.2.3. *The construction given in Ch. 4 is worst case optimal.*

We show a further property of the language each worst-case expression denotes. Recall that we have shown the relative unary complexity of a regular language is bounded by unity (Thm. 3.2.11). Moreover, this bound tight, at least for languages over a growing alphabet (Lem. 3.2.12). We show that this also holds for each $\mathrm{L}(\mu_n)$, which is defined over a fixed alphabet. We argue similar as in the proof of Thm. 5.2.2, by considering a block-maximal word in $w \in \mathrm{L}(\mu_n)$.

Lemma 5.2.4. $\omega(\mathrm{L}(\mu_n)) = 1$.

Proof. First we show that the alphabetic complexity of $\mathrm{L}(\mu_n)$ is $5n$. Recall that the structure of a block-maximal word $w_{\max} \in \mathrm{L}(\mu_n)$ is

$$w_{\max} = b_{1,j_1} b_{2,j_2} \ldots b_{2n,j_{2n}},$$

where $j_i \in \{1, 2\}$ if i is odd and $j_i \in \{3, 4, 5\}$ if i is even. This allows for $5n$ combinations for the indexing b_{i,j_i} of a block in any $w_{\max}$.

Let ν be any expression that denotes $\mathrm{L}(\mu_n)$, and let $b_{i,j}$ and $b_{n,m}$ be distinct blocks that are allowed in block maximal words of $\mathrm{L}(\mu_n)$. In the case that $j \neq m$, the blocks consist of different letters $x_j \neq x_m$, which certainly originate from distinct letter positions in ν. For $j = m$ follows $i \neq n$, and since the blocks are distinct, at least one other block lies between them in a block maximal word. Let w be such a word in $\mathrm{L}(\mu_n)$, i.e., let

$$w := w_1\, x_j^k\, w_2\, x_j^{k'}\, w_3,$$

where k and k' are positive and w_2 is nonempty. We decompose w differently to write it as

$$w := w_1'\, x_j\, w_2'\, x_j\, w_3,$$

with $w_1' = w_1 x_j^{k-1}$ and $w_2' = w_2 x_j^{k'-1}$. If we assume that the two x_j that stick out in w originate from the same position, we can apply a "pumping argument". We then find

$$w_1' x_j (w_2' x_j)^l w_3 \in \mathrm{L}(\mu_n')$$

for arbitrary $l \in \mathbb{N}$. Since w_2 contains more letters than just x_j, the number of blocks in words from $\mathrm{L}(\nu)$ is unbounded. Then $\mathrm{L}(\mu_n) \neq \mathrm{L}(\nu)$ follows, which proves that any expression that any expression that denotes $\mathrm{L}(\mu_n)$ has at least one letter position for every possible indexing of a block allowed in a block maximal word. Since there are $5n$ such indexings, and the alphabetic width of an expression is bounded by the number of letter positions from below, we arrive at $\alpha(\mathrm{L}(\mu_n)) \geq 5n$. Since $\alpha(\mathrm{L}(\mu_n)) \leq |\mu_n|_{\mathcal{A}} = 5n$ we arrive at the sought intermediate result, that $\alpha(\mathrm{L}(\mu_n)) = 5n$.

Again, let ν be any expression that denotes $\mathrm{L}(\mu_n)$ and reaches the alphabetic complexity, i.e., $|\nu|_{\mathcal{A}} = 5n$. This has several consequences for the structure of ν. First, all the literals in ν are then letters, i.e., $\varnothing, \varepsilon \notin \mathrm{sub}(\nu)$. Moreover, since $\varepsilon \in \mathrm{L}(\mu_n)$, every position is the scope of a unary operator in ν. As every position can be repeated arbitrarily often in $w \in \mathrm{L}(\mu_n)$ every position is in the scope of a star. Since the number of blocks is bounded in $w \in \mathrm{L}(\mu_n)$, no sum is in the scope of a star. By the same reason, no base contains distinct letters, so all positions that occur in a fixed base are x_i-positions, for fixed i.

Suppose now that $|\nu|_\omega < 5n$. It then follows that there are two letter positions that are in scope of the same iterations. As we have argued, these are both x_i-positions, possibly within in a bigger product x_i^l. Removing one of these positions, i.e., replacing x_i^l with x_i^{l-1}, does not alter the denoted language.

However, this yields an expression of alphabetic width $5n-1$, which contradicts $\alpha(\mathrm{L}(\mu_n)) = 5n$. Therefore, $|nu|_\omega \geq 5n$ follows.

Since $|\mu_n|_\omega = 5n$ and $\alpha(\mathrm{L}(\mu_n)) = 5n$, the statement follows. □

6 Series Parallel Loop Graphs

In this chapter the class of series-parallel-loop graphs is defined and investigated. This class generalizes that of arc-series-parallel graphs beyond the acyclic case. The arc-series-parallel graphs are well-known from the work of Valdes et al. [52, 53, 54]. Two important results from their work are the efficient decidability of arc-series-parallel graphs and a characterization of its members by forbidden subgraphs. These properties will be generalized to our new class.

Additional Terminology and Notation A *hammock* is a graph G with vertices src and snk, respectively called the *source* and the *sink* of G, s.t. G satisfies the following properties:

1. $\mathrm{d}^-_G(src) = 0$ and $\mathrm{d}^+_G(snk) = 0$,
2. for every $x \in V_G$ there is a (src, x)-path in G, and
3. for every $x \in V_G$ there is an (x, snk)-path in G.

We refer to a hammock G with source src and sink snk as (G, src, snk). The class of hammocks is denoted $\mathbf{H}$. The smallest member of $\mathbf{H}$ is the graph consisting of one vertex and no arcs. Observe that the converse of a hammock (G, src, snk) is the hammock $(G^{\mathrm{R}}, snk, src)$, which means $\mathbf{H}$ is closed under arc-reversal. Thus we may — and usually will — resort to the principle of directional duality for proving properties of $\mathbf{H}$.

If (G, src, snk) contains vertices x and y s.t. x lies on every (src, y)-path in G, then x *dominates* y, while y *is dominated* by x. Symmetrically, if x lies on every (y, snk)-path in G, then x *co-dominates* y. If x dominates and co-dominates y, we say that x *guards* y, resp. that x is a *guard* of y. More generally, if F is a subgraph of G and x dominates, co-dominates, or guards each vertex of G, then x dominates, co-dominates, or guards G, respectively.

The domination and the co-domination relations each induce a partial order on the vertices of a hammock. In the following, this is stated only for the domination relation, but follows symmetrically for co-domination.

Proposition 6.0.5. *Let H be a hammock and let x, y, and z be vertices of H. Then the following properties hold in H:*

1. *Reflexivity: Each vertex dominates itself.*
2. *Transitivity: If x dominates y and y dominates z, then x dominates z.*
3. *Antisymmetry: If x dominates y and y dominates x, then $x = y$.*

Proof. Let src denote the source vertex of H. Showing reflexivity is trivial, as each $x \in V_H$ lies on every (src, x)-path. For transitivity, assume that x lies on every (src, y)-path, and that y lies on every (src, z)-path. Then x lies on the (src, y)-segment of every (src, z)-path and thus on every (src, z)-path. To show antisymmetry, assume that x lies on every (src, y)-path and that y lies on every (src, x)-path. For $x \neq y$, this implies that each (src, x)-path passes through x before reaching x, which is absurd. Thus follows $x = y$. □

Proposition 6.0.6. *Let H be a hammock with source src and distinct vertices x and y. Then exactly one of the following properties hold in H:*

1. *x dominates y*
2. *y dominates x*
3. *Some vertex z dominates both x and y, and H contains a (src, z)-path, a (z, x)-path and a (z, y)-path, which are pairwise internally disjoint.*

Proof. Assume that x dominates y. Since we assumed $x \neq y$, and the domination relation is antisymmetric, it follows that y does not dominate x. Next, let z be any vertex dominating both x and y. This implies that no (src, z)-path passes through x. But since x dominates y by assumption, x must lie on every (z, y)-path. Thus no pair of (z, x)- and (z, y)-paths is internally disjoint.

If y dominates x, the two other properties are excluded by analogous reasoning.

If neither vertex dominates the other, let $z_0, z_1, \ldots, z_n$ denote the vertices that dominate both x and y in H. Such vertices exist, since at least src dominates both x and y. The vertices z_i are linearly ordered by the domination relation: either z_i dominates z_j or vice versa. We may thus assume that the z_i are indexed with increasing distance from src, i.e., $z_0 = src$, and any (z_i, x)- or (z_i, y)-path contains no z_j for $j < i$. Let P_x denote a (z_n, x)-path which does not pass through y; since y does not dominate x by assumption, a path with that property exists. If H contains a (z_n, y)-path that is internally disjoint with P_x, the claim follows for $z = z_n$. Otherwise, every (z_n, y)-path shares an internal

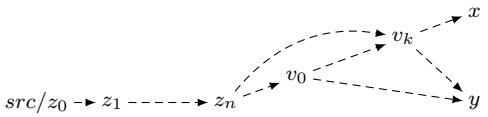


Figure 6.1: Construction in the proof of Prop. 6.0.6.

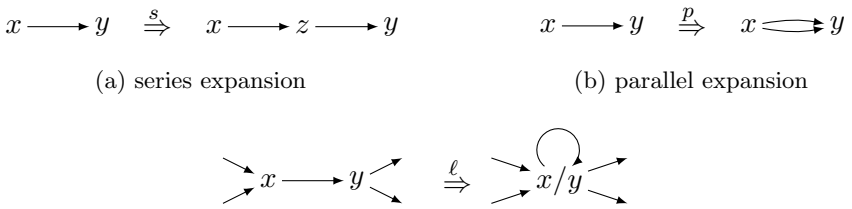
vertex with P_x. Let $v_0, \ldots, v_k$ denote the vertices on P_x that intersect with the (z_n, y)-paths, and assume that the v_i are indexed s.t. P_x contains a segment from v_i to v_{i+1} for each i. If there is only one intersection vertex, namely v_0, this contradicts our choice of z_n, since in that case, v_0 dominates x and y, but is farther from src than z_n. So assume that there are several such vertices and consider those with minimal and maximal index, v_0 and v_k. By a similar contradiction argument as before, we find that there is at least one (z_n, y)-path P_y that passes through v_0, but not v_k. Likewise, some (z_n, y)-path P'_y passes through v_k but not v_0. We construct a (z_n, x)-path Q_x and a (z_n, y)-path Q_y from P_x, P_y, and P'_y, in a way s.t. Q_x and Q_y are internally disjoint. The path Q_x consists of the (z_n, v_k)-segment of P'_y followed by the (v_k, x)-segment of P_x. The path Q_y consists of the (z_n, v_0)-segment of P_x, followed by the (v_0, y)-segment of P_y. □

6.1 Definition and Decidability

Definition 9 (Expansion)**.** The relations $\overset{s}{\Rightarrow}$, $\overset{p}{\Rightarrow}$ and $\overset{\ell}{\Rightarrow}$, called *series expansion*, *parallel expansion* and *loop expansion* respectively, are defined on **H** as follows. Let (G, src, snk) be a hammock with arc $a = xy$, then

- $G \overset{s}{\Rightarrow} H$, if H is obtained from G by replacing a with an (x, y)-path of length two. This is equivalent to removing a, adding a new vertex z and arcs xz and zy. Formally, $H = ((G \setminus a) + z) \cup \{xz, zy\}$.
- $G \overset{p}{\Rightarrow} H$, if H is obtained from G by adding a further xy-arc to A_G. Formally, $H = G \cup xy$.
- $G \overset{\ell}{\Rightarrow} H$, if a is a constriction s.t. $x \neq src$ and $y \neq snk$, and H is obtained from G by merging x and y. Formally, $H = G[x = y]$.

These relations are abbreviated as s-, p- and ℓ-expansion, for series-, parallel- and loop-expansion, respectively. If $G \overset{c}{\Rightarrow} H$ for $c \in \{s, p, \ell\}$ then we say that G



(a) series expansion

(b) parallel expansion

(c) loop expansion, for a constriction xy

Figure 6.2: Changes in a graph upon expanding an xy-arc.

is c-expanded to H and that H is an c-expansion of G. The local changes in a hammock upon expansion are sketched in Fig. 6.2. We may omit the particular type of expansion and write just $G \Rightarrow H$; to this end, we set

$$\Rightarrow := \overset{s}{\Rightarrow} \cup \overset{p}{\Rightarrow} \cup \overset{\ell}{\Rightarrow} .$$

Observe that $\overset{s}{\Rightarrow}$, $\overset{p}{\Rightarrow}$, and $\overset{\ell}{\Rightarrow}$, are indeed relations on $\mathbf{H}$, i.e., if $G \Rightarrow H$ for some hammock G, then H is a hammock with the same source and sink as G. For ℓ-expansion, this property is "enforced" by requiring that the expanded arc is not incident to the source or sink.

Definition 10. The class of *series parallel loop graphs*, denoted **SPL** , is generated by $\overset{s}{\Rightarrow}$, $\overset{p}{\Rightarrow}$ and $\overset{\ell}{\Rightarrow}$ from P_1. It is the smallest class of graphs that satisfies

- $\mathsf{P}_1 \in \mathbf{SPL}$
- If $G \in \mathbf{SPL}$ and $G \overset{s}{\Rightarrow} H$, $G \overset{p}{\Rightarrow} H$, or $G \overset{\ell}{\Rightarrow} H$, then also $H \in \mathbf{SPL}$.

For brevity we speak of *spl-graphs* only. The graph P_1 is called the *axiom* of **SPL**. An example for the construction of an spl-graph from the axiom is shown in Fig. 6.3.

The set of *acyclic* spl-graphs, which consists of those spl-graphs that are generated from P_1 by $\overset{s}{\Rightarrow}$ and $\overset{p}{\Rightarrow}$ is called the class of *series parallel graphs* and denoted **SP**. This class is well-known by now; it was investigated by Valdes et al. [52, 53, 54]. In this and the subsequent chapters, we will come back to the results by these authors at several times. Their results about sp-graphs will be generalized to spl-graphs, while at the same time, we omit any proof for the acyclic case and refer to these previous works.

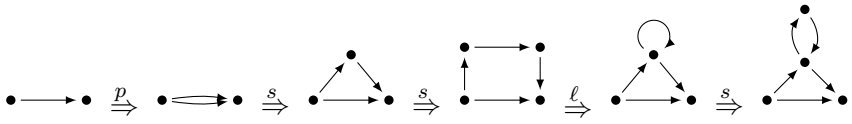


Figure 6.3: Construction of an spl-graph from P_1 by a sequence of expansions.

It should be noted that the recursive definition of **SP** given by Valdes et al. differs from the one we use: instead of replacing arcs — as we do for s- and p-expansions — in the constructive step, they combine two sp-graphs by merging the sources and sinks in different ways. The two definitions can be shown to be equivalent by a straightforward inductive argument, which is omitted here. Let us further mention that the class we refer to as "series parallel graphs" is termed "arc series parallel" in the original works. There is, in fact, a second class of acyclic graphs defined by some means of series and parallel operations, namely, that of "vertex series parallel graphs". These classes are closely related, which is investigated in [52]. Nevertheless, their distinction is irrelevant for us, so we drop the modifier.

Quite obviously, the class of sp-graphs is a properly contained in that of spl-graphs, since the latter contains graphs with cycles, such as the one shown in Fig. 6.2. According to the discussion preceding Def. 10, we find further that every spl-graph is a hammock. This leads to the inclusions

$$\mathbf{SP} \subsetneq \mathbf{SPL} \subseteq \mathbf{H}.$$

We are first going to show that the latter inclusion is proper. An important property for proving this, as well as many further properties of spl-graphs, is the fact that the connectivity of two vertices is invariant under expansion, as long as the considered vertices are not removed.

Proposition 6.1.1. *Let G and H be graphs s.t. $G \Rightarrow H$ and $x, y \in V_G \cap V_H$. Then G contains an (x, y)-path iff H does.*

A first property that relies on this fact is the following:

Proposition 6.1.2. *Every cycle in an spl-graph is guarded by exactly one of its vertices.*

Proof. The claim is vacuously true for P_1, so suppose it is true for $G \in \mathbf{SPL}$ and let $G \Rightarrow H$. It is easy to see that the claimed property carries over if H is derived from G by means of s- or p-expansion.

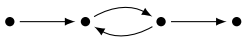


Figure 6.4: Hammock which does not belong to **SPL**

For the case $G \overset{\ell}{\Rightarrow} H$ let $a = xy$ be the arc that is expanded in G and let z denote the merge vertex of x and y in H. Now consider the cycles in H. The cycle that was introduced by expansion consists of z and the loop $l = zz$. This cycle is certainly guarded by its only vertex and thus satisfies the claim. For each other cycle in H we find a corresponding cycle in G. Let v and w be two vertices that are not incident to a in G, which implies that $v, w \in V_G \cap V_G$. Notice that the source and the sink of G are unaffected by expansion, i.e., the source, resp. the sinks, of G and H coincide. Following Prop. 6.1.1, v guards w in H iff v guards w in G. Applied to cycles of H this carries the inductive assumption from G over to H. □

Corollary 6.1.3. SPL $\subsetneq$ H

Proof. We already mentioned that **SPL** $\subseteq$ **H**, which is formally shown by straightforward induction but omitted here. To see that the inclusion is strict, consider the hammock shown in Fig. 6.4 which contains a cycle where neither vertex guards the other. Thus, that hammock defies Prop. 6.1.2 and is therefore no member of **SPL**. □

Deciding whether G is an spl-graph can be done by finding a sequence of expansions from P_1 to G. This will be done by means of a second rewriting system on hammocks, within which we construct such a sequence backwards. In the following definition, recall that a vertex x is simple if $\mathrm{d}^-(x) = \mathrm{d}^+(x) = 1$.

Definition 11. The relations $\overset{s}{\Leftarrow}$, $\overset{p}{\Leftarrow}$ and $\overset{\ell}{\Leftarrow}$, called *series reduction*, *parallel reduction* and *loop reduction* respectively, are defined on **H** as follows. Let (G, s, t) be a hammock, then

- $G \overset{s}{\Leftarrow} H$, if $y \in V_G$ is simple, with predecessor x and successor z, and H is obtained from G by removing y and adding an arc from x to z. Formally, $H = (G - y) \cup xz$.

- $G \overset{p}{\Leftarrow} H$, if G contains parallel arcs a and a' and H is obtained by removing one of them. Formally, $H = G \setminus a$.

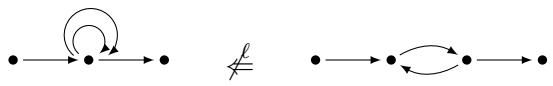


Figure 6.5: Effect of "reducing" a loop on sight. Although the left-hand side is an spl-graph, the right-hand side cannot be reduced to P_1.

- $G \overset{\ell}{\Leftarrow} H$, if G contains an x-loop l s.t. x does not guard any vertex besides itself, no arc is parallel to l, and H is obtained from G by removing l and then splitting x. Formally, $H = (G \setminus l) \ll x \gg$.

As with expansions, we find that the reductions do not destroy the defining properties of a hammock, i.e., these rules are also relations on $\mathbf{H}$. In accordance with the spl-expansions we abbreviate the reduction relations as s-, p- and ℓ-reduction. These relation constitute the rules of the reduction ARS

$$\mathfrak{R} := \langle \mathbf{H}, \overset{s}{\Leftarrow}, \overset{p}{\Leftarrow}, \overset{\ell}{\Leftarrow} \rangle.$$

We further set

$$\Leftarrow := \overset{s}{\Leftarrow} \cup \overset{p}{\Leftarrow} \cup \overset{\ell}{\Leftarrow}$$

and write just $G \Leftarrow H$ if the particular type of reduction from G to H is irrelevant.

Observe that ℓ-reduction is defined for loops on vertices that are slightly restricted. In particular, the restriction ensures that a loop is not ℓ-reduced in the presence of parallel loops. The latter might lead to false negatives, as is shown in Fig. 6.5 by example. In a case like this, the restriction enforces that parallel reduction is applied before loop reduction.

It is important to realize that the relations $\Leftarrow$ and $\Rightarrow$ are not proper duals. This is due to the restrictions we set for ℓ-reduction. To stay with the previous example, consider Fig. 6.5 again. As we remarked just before, the left hand side can not be ℓ-reduced to the right hand side. However, the hammock on the right *can* be ℓ-expanded to the one on the left.

On the other hand, it follows easily from the respective definitions that s- and p-reduction are the duals of s- and p-expansion. Altogether, the immediate relationship between the single expansions and reductions is as follows:

$$\overset{s}{\Leftarrow} = \left(\overset{s}{\Rightarrow}\right)^{-1} \text{ and } \overset{p}{\Leftarrow} = \left(\overset{p}{\Rightarrow}\right)^{-1}, \text{ whereas } \overset{\ell}{\Leftarrow} \subsetneq \left(\overset{\ell}{\Rightarrow}\right)^{-1}.$$

Because of this asymmetry wrt. "loop-operations", there is actually something to prove in order to get the following result.

Proposition 6.1.4. *$G \in \mathbf{SPL}$ iff P_1 is an $\mathfrak{R}$-normal form of G.*

Proof. According to the above, the relationship between expansion and reductions is $\Leftarrow\subsetneq \Rightarrow^{-1}$. Thus, if G can be reduced to the axiom, $G \Leftarrow^\star \mathsf{P}_1$, it also can be constructed from the axiom by expansion, $\mathsf{P}_1 \Rightarrow^\star G$. It follows that $G \in \mathbf{SPL}$.

The converse direction is proven by induction on the structure of G. For $G = \mathsf{P}_1$ the claim certainly holds. Assume that the claim holds for $G \in \mathbf{SPL}$ and consider $H \in \mathbf{SPL}$, derived via $G \Rightarrow H$. For $G \overset{s}{\Rightarrow} H$ and $G \overset{p}{\Rightarrow} H$, we find $H \overset{s}{\Leftarrow} G$ and $H \overset{p}{\Leftarrow} G$ immediately. Since we assumed $G \Leftarrow^\star \mathsf{P}_1$, this also yields $H \Leftarrow^\star \mathsf{P}_1$ in either case. For $G \overset{\ell}{\Rightarrow} H$, let $a = xy$ denote the constriction that is ℓ-expanded, i.e., assume $H = G[x = y]$. Further let z denote the merge vertex of x and y. We need to show that ℓ-reduction is applicable to the z-loop in H, resp. that z does not guard any arc besides that loop or any other vertex in H.

For the sake of contradiction, first suppose that z guards some other vertex. Then, in particular, some vertex k that is guarded by z and the two vertices lie on a common cycle in H. Following Prop. 7.2.8, this cycle is guarded by exactly one of its vertices, since $H \in \mathbf{SPL}$. Since z already guards k, it further guards the whole cycle. But then, it follows that the respective cycle in G is guarded by x and y, which both lie on the cycle. This contradicts Prop. 7.2.8. Therefore, z does not guard any vertex but itself, so $G \overset{\ell}{\Leftarrow} H$ holds.

We have found that $G \Rightarrow H$ implies $H \Leftarrow G$ in each case. Since we assumed $G \Leftarrow^\star \mathsf{P}_1$, this further yields $H \Leftarrow^\star \mathsf{P}_1$, and the inductive step is complete. □

So we can test membership in **SPL** by finding a reduction sequence of a graph to the axiom of **SPL**. In the remainder of this section we show that no particular strategy is necessary in order to do so, due to unique normal forms of the ARS $\mathfrak{R}$. This property is crucial for the efficient decidability of **SPL**. The first step of this argument is to show that an $\mathfrak{R}$-normal form is always reached by a finite number of arbitrary rewriting steps.

Proposition 6.1.5. *The system $\mathfrak{R}$ is terminating.*

Proof. Let $p(H)$ denote the number of arcs and loops in $H \in \mathbf{H}$, i.e., set

$$p(H) := |A_H| + |\{l \mid l \in A_H \text{ and } t_H(l) = h_H(l)\}|.$$

Let $H \Leftarrow H'$ and compare $p(H')$ with $p(H)$, depending on the applied reduction. For s-reduction we find $p(H') = p(H) - 1$, as two arcs are traded for one. For

p-reduction we find $p(H') = p(H) - 1$ or $p(H') = p(H) - 2$, depending on whether a proper arc or a loop is removed. This distinction comes with the property that loops are counted twice in $p(H)$, resp. $p(H')$. We therefore also find $p(H') = p(H) - 1$ for ℓ-reduction, as the loop that is removed from H' weighs heavier than the arc that is introduced in H'.

Thus for a rewriting $H = H_0 \Leftarrow H_1 \Leftarrow H_2 \Leftarrow \cdots$, the value of $p(H_i)$ is strictly decreasing. Since $p(H_i) \in \mathbb{N}$, no infinite rewritings are possible. □

Lemma 6.1.6. *The system $\mathfrak{R}$ is locally confluent.*

Proof. We take the common approach by showing that $G \overset{c_1}{\Leftarrow} H_1$ and $G \overset{c_2}{\Leftarrow} H_2$ imply the existence of a hammock J with $H_1 \Leftarrow^\star J$ and $H_2 \Leftarrow^\star J$ for every combination $c_1, c_2 \in \{s, p, \ell\}$. For each case, such a hammock is explicitly given.

The spl-reductions are expressed according to Def. 11, in terms of elementary operations on graphs and splits of vertices. Recall the properties of elementary operation compiled in Props. 2.3.1, 2.3.2, and 2.3.3, which basically state that the order of elementary operations can be swapped if the added / removed elements are not adjacent or incident. We do not explicitly refer the particular proposition in each case; it will always be clear from the context which property is used.

We first consider the cases where $c_1 = c_2$. If both reductions are p-reductions, $G \overset{p}{\Leftarrow} H_i$, then $H_i = G \setminus a_i$, and either H_i reduces to $J = G \setminus \{a_1, a_2\}$. Next, assume that both reduction are s-reductions, $G \overset{s}{\Leftarrow} H_i$, hence $H_i = (G - y_i) \cup x_i z_i$. If y_1 and y_2 are not adjacent in G, then $y_1 \notin \{x_2, z_2\}$ and $y_2 \notin \{x_1, z_1\}$. In this case, we set $J = (G - \{y_1, y_2\}) \cup \{x_1 z_1, x_2 z_2\}$ to find $H_1 \overset{s}{\Leftarrow} J$, due to

$$\begin{aligned}(H_1 - y_2) \cup x_2 z_2 &= (((G - y_1) \cup x_1 z_1) - y_2) \cup x_2 z_2 \\ &= (((G - y_1) - y_2) \cup x_1 z_1) \cup x_2 z_2 \\ &= (G - \{y_1, y_2\}) \cup \{x_1 z_1, x_2 z_2\} = J,\end{aligned}$$

and symmetrically $H_2 \overset{s}{\Leftarrow} J$. If, on the other hand, y_1 and y_2 are adjacent, assume that G contains an $y_1 y_2$-arc (the converse case is symmetric). Since either y_i is simple, this implies $y_1 = x_2$ and $y_2 = z_1$, as sketched below:

$$x_1 \longrightarrow \underset{=x_2}{y_1} \longrightarrow \underset{=z_1}{y_2} \longrightarrow z_2$$

Thus in particular $H_1 = (G - y_1) \cup x_1y_2$. We set $J = (G - \{y_1, y_2\}) \cup x_1z_2$ and find $H_1 \overset{s}{\Leftarrow} J$ for some s-reduction in y_2. Taking the renaming of vertices into account, this formally is:

$$\begin{aligned}(H_1 - y_2) \cup x_1z_2 &= (((G - y_1) \cup x_1y_2) - y_2) \cup x_1z_2 \\ &= ((G - y_1) - y_2) \cup x_1z_2 = (G - \{y_1, y_2\}) \cup x_1z_2 = J,\end{aligned}$$

and a symmetric argument shows $H_2 \overset{s}{\Leftarrow} J$.

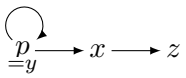
If both reductions are ℓ-reductions, let $l_i = x_ix_i$ denote the loop that allows for $G \overset{\ell}{\Leftarrow} H_i$. If $x_1 = x_2$, these loops are identical, since parallel loops do not admit l-reduction. In this case, we find $H_1 = H_2 = J$, which satisfies the claim trivially. Otherwise, we consider the exposition of H_i in terms of elementary operations, which is

$$H_1 = (G \setminus l_1) \ll x_1 \gg = G \setminus Out(x_1) + x_1' \cup x_1x_1' \cup \{x_1'y \mid xy \in A_G \setminus l\}$$

Now we consider the mixed combinations, $c_1 \neq c_2$. First let $G \overset{s}{\Leftarrow} H_1$ and $G \overset{p}{\Leftarrow} H_2$, say, $H_1 = (G - x) \cup yz$ and $H_2 = G \setminus a$. Then, each vertex incident to a has in- or outdegree at least two, while x is simple. Therefore, a and x are not incident, hence the two reductions can be applied in either order. Formally, this reads as

$$\begin{aligned}H_1 \setminus a &= ((G - x) \cup yz) \setminus a \\ &= ((G - x) \setminus a) \cup yz \\ &= ((G \setminus a) - x) \cup yz = (H_2 - x) \cup yz.\end{aligned}$$

Next, assume $G \overset{s}{\Leftarrow} H_1$ and $G \overset{\ell}{\Leftarrow} H_2$, so $H_1 = (G - x) \cup yz$ again, and $H_2 = (G \setminus l) \ll p \gg$. Observe that $x \neq p$ holds in G, since G is a hammock and does not allow loops at simple vertices. We show that s-reduction of x in H_2 and l-reduction of l in H_1 yield the same graph. Assume that x and p are adjacent, where $p = y$, as sketched below:



We include this renaming in the following formal treatment.

$$\begin{aligned}(H_1 \setminus l) \ll p \gg &= ((G - x \cup yz) \setminus l) \ll p \gg \\ &= ((G - x \cup yz) \setminus l) \ll y \gg \\ &= ((G \setminus l) - x \cup xy) \ll y \gg \\ &= ((G \setminus l) \ll y \gg -x \cup y'z) \ = (H_2 - x) \cup y'z\end{aligned}$$

If finally $G \overset{p}{\Leftarrow} H_1$ and $G \overset{\ell}{\Leftarrow} H_2$, assume the two reductions are specified by $H_1 = G \setminus a$ and $H_2 = (G \setminus l) \ll x \gg$. Observe that a is not an x-loop, since the x-loop l can be reduced. So if $a = yz$, y and z can not both be x. Assume wlog. that $y \neq x$, then we find that either H_i reduces to J, which satisfies

$$(G \setminus l) \ll x \gg) \setminus a = J = ((G \setminus a) \setminus l) \ll x \gg .$$

This completes the proof. □

Corollary 6.1.7. *The system $\mathfrak{R}$ admits unique normal forms.*

Proof. From Prop. 6.1.5 and Lem. 6.1.6 by application of Newman's Lemma. □

Given $G \in \mathbf{H}$, Cor. 6.1.7 allows us to define $\mathrm{R}(G)$, called the *spl-reduct* of G, as the normal form of G in $\mathfrak{R}$. A hammock is called *spl-normal*, or *reduced*, if it coincides with its spl-reduct. The fact that $\mathrm{R}(G)$ is unique strengthens Prop. 6.1.4 considerably.

Theorem 6.1.8. $G \in \mathbf{SPL}$ *iff* $\mathrm{R}(G) = \mathsf{P}_1$.

6.2 Implementation Details

We would like to realize deciding **SPL** based on Thm. 6.1.8, i.e., by implementing spl-reductions on graphs. Recall, however, that the theoretical treatment in Sec. 6.1 is restricted to hammocks. Technically, this is due to ℓ-reduction, which is based on the concept of a guard, which, in turn, is based on the existence of a unique source resp. sink vertex in a graph. Deciding membership of an arbitrary graph G in **SPL** by reductions thus requires knowing whether G is a hammock in the first place, and, if so, what its source and sink are.

To restate the definition, G is a hammock iff there are distinct vertices, src and snk, in G s.t. $\mathrm{d}^-(src) = 0$, $\mathrm{d}^+(snk) = 0$, and every vertex of G lies on

a (src, snk)-path. Observe that these conditions imply that src and snk are unique; moreover, they are either distinct, or the hammock consists of a single vertex and no arcs. This leads to Alg. 2, which accepts a hammock by returning its source and sink, and rejects any other graph.

Algorithm 2: Test whether a graph is a hammock and return the source and sink vertex in the positive case.

Input: Graph G
Output: (src, snk) if $G \in \mathbf{H}$, false otherwise
$src \leftarrow$ NULL
$snk \leftarrow$ NULL
foreach $v \in V_G$ **do**
  if $\mathrm{d}^-(v) = 0$ **then**
    if $src =$ NULL **then** $src \leftarrow v$ **else** return false
  if $\mathrm{d}^+(v) = 0$ **then**
    if $snk =$ NULL **then** $snk \leftarrow v$ **else** return false
if $src =$ NULL *or* $snk =$ NULL **then** return false
$R_1 \leftarrow \mathrm{reach}(src, G)$
$R_2 \leftarrow \mathrm{reach}(snk, G^{\mathrm{R}})$
if $|R_1| \neq |V_G|$ *or* $|R_2| \neq |V_G|$ **then** return false **else** return (src, snk)

Proposition 6.2.1. *Membership in* $\mathbf{H}$ *can be decided linear time.*

Proof. We analyze the behavior of Alg. 2 on input G. First, G is scanned for a possible source and sink by checking the in- and out-degree of each vertex. Since the source and sink of a hammock are unique, G is rejected if several vertices with the degree characteristic of either the source or the sink are found.

If no candidate for the sink or the source is found upon scanning G, the algorithm rejects, too.

Next, if unique distinct candidates for the source and the sink were found, the algorithm checks whether each vertex of G lies on a path between those vertices. This is the obviously case iff every vertex of G can be reached from src, and snk can be reached from every vertex. The vertices reached from src are found by a depth-first search starting in src. Symmetrically, the vertices that reach snk in G are the ones which are reached from snk in G^{R}; they are computed accordingly. If every vertex is reached from src and reaches snk, the input G is accepted as a hammock by returning its source and sink; otherwise, G is rejected. The algorithm certainly terminates and thus decides $\mathbf{H}$.

The loop scanning V_G for possible source and sink vertices obviously runs in time $\mathcal{O}(|V_G|)$. The sets R_1 and R_2 can be found by depth-first searches on G, which require $\mathcal{O}(|V_G| + |A_G|)$ steps each. The overall running time of the algorithm is thus $\mathcal{O}(|V_G| + |A_G|)$, which is linear in the size of G, as stated. □

According to Prop. 6.2.1, a single test whether a given graph is a hammock adds no asymptotically relevant computational overhead[1]. In the following, we thus assume to operate on hammocks.

A straightforward approach to decide membership in **SPL** would be to simply apply reductions as long as possible. Following Cor. 6.1.7, this procedure terminates in the unique normal form of the input. According Thm. 6.1.8, this normal form equals the axiom iff the input hammock is in **SPL**.

We will follow a more structured approach, mainly for the sake of easening the run-time analysis. Let us mention once more that spl-reduction is locally confluent. This allows us to pursue any reduction sequence we see fit, as long as it is exhaustive, since the resulting normal form is independent of the particular rewriting that lead to it. We start with stating some observations about spl-reductions, which motivate the reduction strategy pursued in the presented algorithm.

In the following propositions we consider how p-reduction might follow after s- or ℓ-reduction of a p-normal graph.

Proposition 6.2.2. *Let G be p-normal and let $v \in V_G$ be simple with predecessor x and successor y. Assume $G \overset{s}{\Leftarrow} H$ due to s-reduction of G in v. Then H is p-normal iff $xy \notin A_G$.*

Proof. Reducing G to H removes some elements from H and introduces an xy-arc. Now if H contains parallel arcs, whereas G does not, these must be xy-arcs. Thus follows the claim. □

Proposition 6.2.3. *If G is p-normal and $G \overset{\ell}{\Leftarrow} H$, then H is p-normal.*

Proof. Let $G \in \mathbf{H}$ be p-normal, and let G contain an x-loop that allows for loop reduction to H. Let x_1 and x_2 denote the split-vertices of x and let $a = x_1x_2$ be the arc introduced by reduction. By definition, a is the only x_1x_2-arc in H, i.e., no arc is parallel to a. Every other arc of H is either also in G, or derived

[1] To be precise, this is true only if the computations to follow are not all sublinear.

from an arc of G by replacing the tail or head x with x_1 resp. x_2. Since G is free of parallel arcs, thus H is free of parallel arcs, too. □

As far as ℓ-reduction is concerned, we need to decide for each loop if ℓ-reduction is applicable. Recall that an x-loop can be reduced in (G, s, t) iff there is no parallel x-loop and x does not guard any other vertex. Checking for the presence of a parallel loop is straightforward. Testing if a vertex is a guard can be done in linear time by testing for the properties that define a guard.

Proposition 6.2.4. *Let x be a vertex of the hammock G. Testing if x is a nontrivial guard takes time $\mathcal{O}(|A_G|)$.*

Proof. Let *src* and *snk* denote the source and sink of G, respectively. Recall that x guards a vertex y if x lies on every (src, y)-path and every (y, snk)-path. This is the case iff the graph $G - x$ contains no (src, y)- and (y, snk)-paths at all. Since we check for a nontrivial guard, we implicitly assume $x \neq y$.

As before, let $\mathrm{reach}(v, K)$ denote the set of vertices reached from v in the graph K. For our purpose, it is sufficient to test if y is contained in $\mathrm{reach}(src, G - x)$ or $\mathrm{reach}(snk, (G - x)^{\mathrm{R}})$. If y is contained in neither set, then x guards y in G.

The sets $\mathrm{reach}(src, G-x)$ or $\mathrm{reach}(snk, (G-x)^{\mathrm{R}})$ are again found by depth-first searches in the corresponding graphs. The time required for each search is $\mathcal{O}(|V_{G-x}| + |A_{G-x}|)$, which is bounded by $\mathcal{O}(|V_G| + |A_G|)$. Since G is assumed to be a hammock, it is connected, thus the latter term is bounded by $\mathcal{O}(|A_G|)$ in turn. Thus follows the statement. □

Our method to decide membership of a hammock in **SPL** is based on that described by Valdes et al. [54] for **SP**. A significant difference — other than we obviously have to provide the means for ℓ-reduction — is that we rearrange the interplay of s- and p-reductions. In a first step, we convert the input to its p-normal form; the pseudocode for this part is given in Alg. 3. Following that, we convert this p-normal graph to its spl-normal form, which is also the spl-normal form of the initial graph. The latter reduction makes use of Props. 6.2.3 and 6.2.2, i.e., p-reduction only takes place after a certain case of s-reduction. This "trick" was applied before, by Schoenmakers [46], for the recognition of **SP**. The pseudocode for this second step is provided in Alg. 4.

Proposition 6.2.5. *The p-normal form of $G \in \mathbf{H}$ is computed by Alg. 3 in time $\mathcal{O}(|A_G|^2)$.*

Algorithm 3: Construct the p-normal form of a hammock

Input: hammock (G, src, snk)
Output: p-normal form of G

```
foreach v ∈ A_G \ snk do
    aList ← v.out_arcs()
    while aList ≠ ∅ do
        a ← aList.pop()
        foreach a' ∈ aList do
            if h(a) = h(a') then
                aList ← aList \ a'
                G ← G \ a'
```

Proof. Every arc of G leaves some vertex, except for the sink of G, so it suffices to consider all out-arcs. We show that after one iteration of the main loop, G is free of parallel arcs that are out-adjacent to v, the vertex chosen at the beginning of this iteration.

Initially, $aList$ contains all out-arcs of v. The while-loop of the procedure removes an arc from $aList$, say $a = vx$. Any other out-arc of v and with head x is then removed from $aList$ and G, thus at the end of the innermost loop, G contains only one vx-arc. This is repeated for all out-neighbors of v and, in the main loop, for every v. Thus the procedure terminates in a p-normal graph.

For a fixed v, reducing all parallel out-arcs requires at most

$$\sum_{1 \leq i \leq \mathrm{d}^+(v)} i = \mathcal{O}(\mathrm{d}^+(v)^2)$$

operations, as the first out-arc is compared to all other out-arcs, the second out-arcs is compared to all except the first one, etc. For all vertices, the overall number of operations is bounded from above by

$$\sum_{v \in V_G} \mathcal{O}(\mathrm{d}^+(v)^2) = \mathcal{O}(\sum_{v \in V_G} \mathrm{d}^+(v)^2) = \mathcal{O}(|A_G|^2).$$

□

We separately prove that algorithm 4 is correct and that its running time is cubic in the number of vertices of the input graph.

Algorithm 4: Construct the spl-normal form of a p-normal hammock

Input: p-normal (G, src, snk)
Output: spl-normal form of G

$C \leftarrow V_G \setminus \{src, snk\}$
while $C \neq \emptyset$ **do**
  $v \leftarrow C.choose()$
  $C.remove(v)$
  if $\mathrm{d}^-(v) = \mathrm{d}^+(v) = 1$ **then**
    $x \leftarrow first\_predecessor(v)$
    $y \leftarrow first\_successor(v)$
    if $xy \in A_G$ **then**
      $G \leftarrow (G - v)$
      if $x \neq src$ **then** $C.insert(x)$
      if $y \neq snk$ **then** $C.insert(y)$
    else
      $G \leftarrow (G - v) \cup xy$
  if $l = vv \in A_G$ **then**
    $R_1 \leftarrow reach(src, G - v)$
    $R_2 \leftarrow reach(snk, G^{\mathrm{R}} - v)$
    if $R_1 \cup R_2 = V_G$ **then**
      $G \leftarrow (G \setminus l) \ll v \gg$
      $C.insert(v)$
      $C.insert(v')$

Proposition 6.2.6. *Let G be a p-normal hammock. Then Alg. 4 computes the spl-normal form of G.*

Proof. First, we show that the algorithm is correct, i.e., that it terminates in the spl-normal form of its input. To this end we track the set C of vertices that are candidates for reduction. We find that at all times, C contains all vertices that allow for s-reduction or that carry a loop which allows for l-reduction in the current G. In addition we show that G remains p-normal. Formally, we prove by induction that these two properties are a loop invariant, i.e., that they hold before entering and after leaving the body of the loop.

This is certainly true when the body is entered the first time. First, G is p-normal by specification of Alg. 4. Second, C initially contains all vertices of G except the source and the sink, therefor it certainly contains every vertex that can possibly be reduced.

Assume the claim is true when the loop is entered, and let v be the vertex that is removed from C. Notice that v may not be simple and carry a loop at the same time. Thus, exactly one of the following cases applies:

1. If v is neither susceptible to s- or ℓ-reduction, then G is unaltered, i.e., it stays p-normal. Moreover, every vertex that does allow for reduction is already in C by assumption.

2. If v is simple, it is removed in either subcase and needs not be minded anymore. Consider any vertex $z \notin C \cup \{x, y\}$, i.e., one that is not adjacent to v. Clearly, s-reduction in v does not change the in- or out-degree of z, so we do not miss a vertex that becomes susceptible to s-reduction due to the changes just applied. Following Prop. 6.1.1, reachability of or from z relative to any other vertex in G is invariant under reduction, thus we neither miss a vertex that carries a loop and becomes a non-guard, i.e., susceptible to l-reduction.

   We know from Prop. 6.2.2 that s-reduction of a p-normal graph introduces a parallel pair of arcs iff one of these arcs is present before reduction. In that case, this is the only parallel pair, so a single p-reduction reinstates that the graph we work on is p-normal. Formally, this sequence of reductions is $((G - v) \cup a) \setminus a'$, where both a and a' are xy-arcs. Observe that the resulting graph is isomorphic to $G - v$. We thus "simulate" the follow-up p-reduction by just removing v. On the other hand, if x and y are not adjacent, s-reduction is carried out properly. These actions realize — or

rather, simulate — sound s-reduction, possibly followed by sound p-reduction. Moreover, the graph remains p-normal.

3. If v carries a loop that can be reduced, the properties for any $z \notin C \cup \{v, v'\}$ follow as in the previous case. On the other hand, since the out-degree of v is altered and v' has not been considered yet, they are added to C. Thus the claimed property of C carries over to the next iteration of the main loop. The fact that G stays p-normal was observed before, in Prop. 6.2.3.

The algorithm eventually terminates, since every vertex that appears at some point of the reduction is only added a finite number of times to C, from which some element is removed with each iteration. Upon termination, G is p-normal, and since C is then empty, G does not contain vertices that can be s- or l-reduced, i.e., G is s- and l-normal, too.

Therefore, the algorithm terminates in the spl-normal form of its input. □

Proposition 6.2.7. *Algorithm 4 runs in time $\mathcal{O}(|V_G|^3)$ on G.*

Proof. We first bound the time required for a single execution of the main loop. Afterwards we bound the overall number of calls of this loop.

As the body of the main loop is free of inner loops, the latter value is the maximum running time among all operations. We find two computations that "compete" as the factor determining asymptotic running time, depending on the structure of G. On the one hand, the time required to compute either R_i by depth-first search is $\mathcal{O}(|A_G|)$, since G is connected. On the other hand, computing the union of R_1 and R_2 runs in $\mathcal{O}(\min\{|R_1|, |R_2|\} \log(|R_1| + |R_2|))$. Since $R_i \subseteq V_G$, this can be relaxed to $\mathcal{O}(|V_G| \log |V_G|)$, and by the same argument, testing $R_1 \cup R_2$ and V_G for equality runs in $\mathcal{O}(|V_G| \log |V_G|)$, too.

Thus if G is dense, $\mathcal{O}(|A_G|)$ determines the time spent at the most in the body, whereas if G is sparse, $\mathcal{O}(|V_G| \log |V_G|)$ does. However, since G is p-normal, $|A_G| \in \mathcal{O}(|V_G|^2)$ follows. Thus, either case is covered by $\mathcal{O}(|V_G|^2)$ as an upper bound to the time spent in one iteration of the main loop.

The number of times that the main loop is executed equals the cumulative number of times each vertex that appears at some point in G is chosen from C. For each vertex, this number is obviously equal to the number of times it is added to C throughout the execution of the algorithm. □

As a corollary we get the main result of this section, stating that the class **SPL** is efficiently decidable.

Theorem 6.2.8. *Membership of G in* **SPL** *can be decided in time*

$$\mathcal{O}(\max\{|V_G|^3, |A_G|^2\}).$$

This result concludes the constructive treatment of **SPL** for now. We will get back to Alg. 4 in Ch. 7 where we consider finite automata that are structurally spl-graphs.

7 Forbidden Minor Characterization

In this chapter we present an alternative characterization of the class **SPL**. The recursive definition given in chapter 6 specifies how to obtain spl-graph constructively; it tells us how a "follow-up" spl-graph may look, starting from an spl-graph that is already known. The second characterization specifies exactly how an spl-graph must *not* look, by excluding a set of structural properties.

An "obstruction set" characterization of a graph class $\mathbf{C}$ consists of two parts: First, there is the obstruction set, which is a set of graphs itself, say, $\mathbf{O} \subseteq \mathbf{G}$. Second, a relation R on $\mathbf{G}$, that models the presence of a substructure in a graph. A characterization of that type reads as follows:

$$\forall G \in \mathbf{G}\, \forall O \in \mathbf{O} : G \in \mathbf{C} \text{ iff } (O, G) \notin R.$$

The usual choices for R are the subgraph or the minor relation. Accordingly, the characterization is called a "forbidden subgraph" characterization or a "forbidden minor" characterization. Either relation might be modified; in particular, the "induced" variant of either relation is often considered. This allows to model the absence of (possibly undirected) arcs in forbidden graphs. Put differently, the induced variant requires that if a forbidden graph is present in some way as a substructure, then this substructure appears inside a larger structure that translates back to a supergraph of the forbidden graph.

There is a great number of classes of *undirected* graphs that admit an obstruction set characterization while being defined by other means. The best known example is probably the class of planar graphs, characterized by Kuratowski as the set of all graphs that do not admit either of two particular minors [33]. A compilation of further examples is presented in the survey by Brandstädt et al. [7].

Informally, an obstruction set characterization for a class of undirected graphs trivially carries over to directed graphs by considering all orientations of each forbidden graph. This is merely a way of restating the same characterization in a different fashion. As far as directed graphs that do not merely reflect undirected graphs are concerned, there is only a handful of results for classes that admit

characterization of this kind along with a second, unrelated characterization; examples are given in [24, 52, 42, 20, 36] [1].

The obstruction set characterization of **SPL** which we present in this chapter is of the forbidden minor type. More precisely it is defined by two relations resembling the notion of *topological* minors, which is well established for undirected graphs (cf. [13, 3]). It is worth noting that there is no general agreement on the concept of a minor on graphs as we consider them, i.e., on directed graphs. Still, all approaches to provide such a notion include the topological variant by definition. This justifies the convention we use in this chapter's treatment, namely to drop the "topological" modifier from the terminology and speak only of "minors".

Additional Terminology and Notation In the following, let 2^G denote the set of subgraphs of G.

Definition 12. An *embedding* e of F in G is a pair of injections $e_V : V_F \to V_G$, $e_A : A_F \to 2^G$, satisfying

1. if $a \in A_F$, then $e_A(a)$ is an $(e_V(t(a)), e_V(h(a)))$-path in G, and
2. if $a, a' \in A_F$ are distinct, then $e_A(a)$ and $e_A(a')$ are internally disjoint.

If an embedding of F in G exists, we call F a *minor* of G and write $F \preccurlyeq G$. If F is no minor of G, we say that G is F-*free*. More generally, if M is a set of graphs s.t. G is F-free for every $F \in M$, we say that G is M-free.

We also say that $F \preccurlyeq G$ is *realized* by e, and for $x \in V_F$ we refer to $e_V(x)$ as the *peg* of x in G wrt. e. The notation for the two maps constituting an embedding is simplified by writing $e(x)$ for $e_V(x)$ and $e(a)$ for $e_A(a)$. It shall always be clear from the context whether an argument is a vertex or an arc, so no confusion will arise.

An embedding e of F defines the graph

$$e(F) := (e_V(V_F) \cup \bigcup_{a \in A_F} V_{e(a)}, \bigcup_{a \in A_F} A_{e(a)}, \bigcup_{a \in A_F} t_{e(a)}, \bigcup_{a \in A_F} h_{e(a)}).$$

Recall that a DF is any graph that is derived from F by successively subdividing arcs. Equivalently, a DF is obtained by replacing arbitrary arcs $a_i = x_i y_i$ with (x_i, y_i)-paths. We find the following:

[1] As a matter of fact, these references constitute all the (nontrivial) examples the author was able to track down in the literature.

Proposition 7.0.9. *Let e be an embedding of F in G. Then $e(F) \subseteq G$ holds, and $e(F)$ is a $\mathrm{D}F$.*

Proof. The fact that $e(F)$ is a subgraph of G follows due to $V_{e(F)} \subseteq V_G$ and $\mathsf{A}_{e(F)} \subseteq A_G$. Moreover, $e(F)$ contains a unique $(e(x), e(y))$-path for every xy-arc of F, and these paths are internally disjoint by definition. □

This implies an equivalent characterization of embeddings, resp. minors, by means of subdivisions and subgraphs. We make frequent use of this fact by switching back and forth between the two characterizations, choosing the notion that better fits our purpose.

Proposition 7.0.10. *The following statements are equivalent:*

1. *F is a minor of G.*
2. *G contains a $\mathrm{D}F$.*

Proof. If F is a minor of G, then $F \preccurlyeq G$ is realized by some embedding e. Then G contains a $\mathrm{D}F$, as shown in Prop. 7.0.9. Conversely, let G contain a fixed $\mathrm{D}F$. Let be $e_V : V_F \to V_G$ the injection that maps $x \in V_F$ to the vertex $x' \in V_G$ that reflects x in this $\mathrm{D}F$. Likewise, let $e_A : A_F \to 2^G$ be the injection that maps $a \in A_F$ to the $(e_V(t(a)), e_V(h(a)))$-path in the $\mathrm{D}F$. Then e_V and e_A constitute an embedding of F in G, so $F \preccurlyeq G$ follows. □

A simple property of embeddings is that the in- and out-degree of a peg does not exceed the respective degree of the peg's preimage.

Proposition 7.0.11. *Let e realize $F \preccurlyeq G$. Then for $x \in V_F$ we find $\mathrm{d}_F^-(x) \leq \mathrm{d}_G^-(e(x))$ and $\mathrm{d}_F^+(x) \leq \mathrm{d}_G^+(e(x))$.*

Proof. For $\mathrm{d}_F^-(x) = k$ the definition of an embedding implies that G contains k internally disjoint paths that terminate in $e(x)$. Since the predecessors of $e(x)$ on any two such paths are distinct, G contains at least k predecessors of $e(x)$, i.e., $k \leq \mathrm{d}_G^-(e(x))$. The symmetric argument shows $\mathrm{d}_F^+(x) \leq \mathrm{d}_G^+(e(x))$. □

A stricter variant of embeddings, resp. minors, and subdivisions, allows to model the absence of arcs in a minor.

Definition 13. Let e be an embedding of F into G and assume $xy \notin A_F$. A *bypass* wrt. e is an $(e(x), e(y))$-path in G that does not pass through a further peg of F. Such a path is also called an (x, y)-bypass of F in G wrt. e. If G contains no bypass of F wrt. e, we call e a *bare embedding* of F in G, and $e(F)$ a *bare* $\mathrm{D}F$ in G. If a bare embedding of F in G exists, we call F a *bare minor* of G and write $F \preccurlyeq_{\mathrm{b}} G$.

Let us stress that a bypass is a path that connects pegs in a way that does not reflect connectivity of the preimages of these pegs in the embedded graph. Even if $xy \notin A_F$, F might well contain an (x, y)-path, passing through other vertices of F. A bypass connects the pegs of x and y in G without respecting their connectivity in F.

7.1 Effects of Expansion

We start our search for an obstruction set by investigating the effects of spl-expansion on the presence, resp. absence, on certain subdivisions that may occur as subgraphs. First, we give an infinite family of excluded minors for spl-graphs. Informally, the members of this class are defined by explicitly prohibiting features that result from an expansion step.

Definition 14. A graph B is called *bulky* if it contains neither parallel arcs nor loops, and every vertex $x \in V_B$ satisfies $\mathrm{d}_B^-(x) \geq 2$ or $\mathrm{d}_B^+(x) \geq 2$. The class of bulky graphs is denoted $\mathbf{B}$.

Notice that a bulky graph does not contain simple vertices. We find that bulky minors, resp. subdivisions, are not introduced by expansion alone, i.e., coming from an otherwise $\mathbf{B}$-free graph.

Lemma 7.1.1. *Assume that G is $\mathbf{B}$-free and let $G \Rightarrow H$. Then H is $\mathbf{B}$-free.*

Proof. Let G be $\mathbf{B}$-free and assume $G \Rightarrow H$. For the sake of contradiction, suppose further that $B \preccurlyeq H$ holds for some $B \in \mathbf{B}$. We consider the three expansions from G to H separately.

1. $G \overset{s}{\Rightarrow} H$: If $B \preccurlyeq H$, a $\mathrm{D}B$ must have emerged from s-expansion. At the most, the effect of s-expansion on a subdivision is to subdivide it further. But since G contains no $\mathrm{D}B$ by assumption, the vertex z, which was introduced upon expansion, must be a peg of some $z' \in V_B$ wrt. an embedding e. However, as

z is simple in G while B, being bulky, is free of such vertices, this contradicts Prop. 7.0.11.

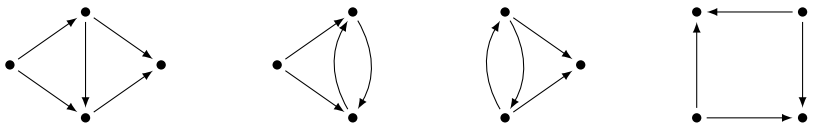
2. $G \overset{p}{\Rightarrow} H$: As in the previous case, assuming $B \preccurlyeq H$ implies that an DB emerges in the process of expanding G to H. Let $a = xy$ denote the expanded arc, which is present in G and H, and let $a' = xy$ be the arc that is introduced with expansion. The only pair of vertices s.t. G and H contain a different number of internally disjoint paths from one vertex to the other, is the pair x, y. Thus $B \preccurlyeq H$ implies that x and y are both pegs of some vertices of B, and that a and a' each represent a (trivial) subdivision of arcs from the preimage of x to the preimage of y. This requires parallel arcs in B, which contradicts that B is bulky.

3. $G \overset{\ell}{\Rightarrow} H$: Again, $B \preccurlyeq H$ implies that some DB was introduced with the expansion. Let $a = xy$ denote the constriction in G that allows ℓ-expansion in the first place and let $l = zz$ be the loop introduced by merging x and y, i.e., let z denote the merge vertex x and y. If e realizes $B \preccurlyeq H$, then $z = e(q)$ for some $q \in V_F$, i.e., z is a peg of B. Otherwise a DB would be present in G already, as implied by Prop. 6.1.1, but contradicting the assumption that G is **B**-free.

   The case continues with a slightly more involved argument than for the previous cases. Consider the graph $H' := H \setminus l$, derived from H by removal of the "new" loop. As the bulky graph B is free of loops, l is no part of the DB in H. Consequently, H' contains the same DB as H does, with $z = e(q)$ (since $V_H = V_{H'}$). The advantage of H' over H is that $\mathrm{d}^-_{H'}(z) = \mathrm{d}^-_G(x)$ and $\mathrm{d}^+_{H'}(z) = \mathrm{d}^+_G(y)$ hold.

   We show that the in-degree of q in B is at least two by rejecting the remaining values. First, suppose $\mathrm{d}^-_B(q) = 0$. Then, only the out-arcs of z belong to $e(B)$ in H'. From $\mathrm{d}^+_{H'}(z) = \mathrm{d}^+_y(G)$ now follows that G contains a DB already, realized by the embedding e' which is defined as e except that $e'_V(q) = y$ and the images for out-arcs of q under e'_A are (y, k)-paths instead of (z, k)-paths. This contradicts the assumption that G is **B**-free. We find $\mathrm{d}^-_B(q) \neq 1$ by a similar argument. If we suppose $\mathrm{d}^-_B(q) = 1$, then the sole path of the DB that enters z in H' can be realized in G (for a slightly different DB, namely, a subdivision of $e(B)$) by using the constriction a. Then $B \preccurlyeq G$ would be realized by e', defined as before, thereby contradicting **B**-freeness of G.

   Thus follows $\mathrm{d}^-_B(q) \geq 2$, and a symmetric argument shows $\mathrm{d}^+_B(q) \geq 2$. Now let B' denote the graph derived from B by splitting q into q_1 and q_2. Then $\mathrm{d}^-_{B'}(q_1) \geq 2$ and $\mathrm{d}^+_{B'}(q_2) \geq 2$, while all other vertices and their degrees are



(a) N (b) C (c) C^{R} (d) Q

Figure 7.1: Crucial subset of bulky graphs, $\mathbf{F} = \{\mathsf{N}, \mathsf{C}, \mathsf{C}^{\mathrm{R}}, \mathsf{Q}\}$.

identical in B and B'. Thus B' is bulky, too. But since H contains a DB it now follows that G contains a DB', contradicting the assumption that G is **B**-free.

□

Corollary 7.1.2. *Every spl-graph is* **B***-free.*

Proof. Clearly, P_1, the axiom of **SPL**, is **B**-free. Assume that $G \in \mathbf{SPL}$ is **B**-free and let $G \Rightarrow H$. Then $H \in \mathbf{SPL}$ is **B**-free according to Lem. 7.1.1. □

For the purpose of characterizing **SPL**, it suffices to consider a finite subset of **B**. The four bulky graphs that are relevant for the sought characterization are gathered in the set

$$\mathbf{F} := \{\mathsf{N}, \mathsf{C}, \mathsf{C}^{\mathrm{R}}, \mathsf{Q}\},$$

which is shown in Fig. 7.1. Observe that membership in **B** is invariant under arc-reversal and that **F** is closed under arc-reversal: C and C^{R} are mutually converse, whereas both N and Q are self-converse.

We have shown in Lem. 7.1.1 that bulky subdivisions are not introduced by expanding an **B**-free graph. In contrast, we find that bulky subdivisions can be *removed* by expansion. This is possible with ℓ-expansion, and happens for **F**.

Proposition 7.1.3.

1. *Let $B \preccurlyeq G$ for some $B \in \mathbf{B}$. If $G \overset{s}{\Rightarrow} H$ or $G \overset{p}{\Rightarrow} H$ holds, then $B \preccurlyeq H$.*
2. *There are graphs G, H, s.t. $G \overset{\ell}{\Rightarrow} H$, and G has a minor in* **B** *while H is* **B***-free.*

Proof.

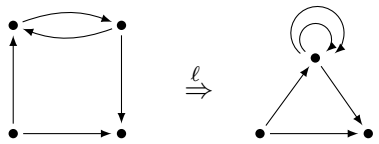
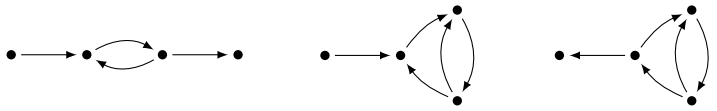


Figure 7.2: The subgraph Q is removed upon loop expansion.



(a) Φ (b) Ψ (c) Ψ^{R}

Figure 7.3: Set of graphs $\mathbf{K} = \{\Phi, \Psi, \Psi^{\mathrm{R}}\}$.

1. Let $B \in \mathbf{B}$ be a minor of G, then G contains an DB as a subgraph. At the most, this subdivision is further subdivided by s-expansion. Clearly, p-expansion does not remove any subgraph at all. Therefore, H contains a DB if $G \overset{s}{\Rightarrow} H$ or $G \overset{p}{\Rightarrow} H$ holds.

2. Subdivisions of the bulky graph Q (cf. Fig. 7.1d), can be removed with ℓ-expansion. This is shown exemplarily in Fig. 7.2 for the trivial subdivision, i.e., the graph Q itself as a subgraph.

□

The graph N is already known in connection with **SPL**, or rather with its acyclic members. Recall that this subclass is denoted **SP**. It was shown by Valdes [52] that **SP** admits a forbidden minor characterization within the class of hammocks by means of N alone.

Theorem 7.1.4 (Valdes)**.** *Let $H \in \mathbf{H}$ be acyclic. Then $H \in \mathbf{SP}$ iff $\mathsf{N} \not\preccurlyeq H$.*

Proof. In Valdes [52, Sec. 4.5] □

We make use of Thm. 7.1.4 whenever acylic graphs are considered. Getting back to the general case, we observe that **F**-freeness — just as **B**-freeness, for that matter — does not suffice to identify a hammock as a member of **SPL**. For example, the hammock shown in Fig. 6.4 is **B**-free. As we have shown in Sec. 6.1, this graph is not contained in **SPL**, since it does not satisfy Prop. 6.1.2.

To get an adequate obstruction set characterization of **SPL**, we resort to bare embeddings of non-bulky graphs. The three additional graphs that will be part of the sought characterization constitute the set

$$\mathbf{K} := \{\Phi, \Psi, \Psi^{\mathrm{R}}\},$$

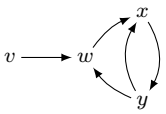
which is shown in Fig. 7.3. Observe that **K** is closed under arc-reversal. We find that spl-graphs are free of bare minors in **K**.

Lemma 7.1.5. *If $G \in \mathbf{SPL}$, then $K \not\preccurlyeq_{\mathrm{b}} G$ for $K \in \mathbf{K}$.*

Proof. We prove the claim for Φ first. To this end we name the vertices of Φ from v to y, as shown:

Let $(G, src, snk) \in \mathbf{SPL}$, and let e realize $\Phi \preccurlyeq G$. Since w and x lie on a cycle in Φ, their pegs $e(w)$ and $e(x)$ lie on a cycle C in G. Then Prop. 7.2.8 states that C is guarded by a unique $g \in V_C$. Because G contains a DΦ, it also contains an $(e(x), e(y))$-path. But since g co-dominates $e(x)$, it also co-dominates $e(y)$: thus G contains an $(e(y), g)$-path P, and since g lies on a cycle with $e(w)$ and $e(x)$, G further contains $(e(y), e(w))$- and $(e(y), e(v))$-paths. However, either path is a bypass, since $yw, yx \notin A_{\Phi}$. Therefore, e is not bare.

The proof is similar for Ψ: again we refer to the vertices of the considered graph individually, as in the sketch below:



As before, let $(G, src, snk) \in \mathbf{SPL}$, and let e realize $\Psi \preccurlyeq G$. Then $e(w)$, $e(x)$ and $e(y)$, lie on a cycle C in G and a unique $g \in V_C$ guards C. Since $e(\Psi)$ is a DΨ, the graph G contains paths from $e(v)$ to $e(w)$, $e(x)$, and $e(y)$. Since g dominates the latter three pegs, it also dominates $e(v)$, i.e., x lies on every $(src, e(v))$-path in G. Thus, G contains a $(C, e(v))$-path, and further, a path from at least one of $e(w)$, $e(x)$, or $e(y)$, to $e(v)$, s.t. this path does not pass through any of the other pegs. Since Ψ contains no in-arc of v, the embedding e is not bare.

For Ψ^{R}, the claim follows immediately from above proof for Ψ and the principle of directional duality. □

(a) $\Phi \preccurlyeq G$ (b) $\Psi \preccurlyeq G'$

Figure 7.4: Members of **SPL** with minors in **K**.

Informally, **K** consists of the prototypical graphs with cycles that do not satisfy Prop. 7.2.8. Adding certain arcs to Φ, Ψ, and Ψ^{R} mends this deficiency. Regarding the bare minor relation, such arcs are are forbidden in the disguise of prohibited bypasses. Examples for spl-graph with minors in **K**, obviously not bare ones, are given in Fig. 7.4.

7.2 Effects of Reduction

In contrast to the expansion relations, we do not study the effects of spl-reductions on the presence, resp. absence, of bulky minors. Instead, we restrict treatment of bulky minors to **F**, which is sufficient for our intentions. The reason for this is, bluntly put, that analyzing reductions is considerably more tedious than analyzing expansions. The main effort comes with loop reduction; this will become apparent in proof of virtually every statement that involves the operation.

First, we consider the (restricted) analogue to Lem. 7.1.1, i.e., the effects of reduction on **F**-free hammocks. We will find that the members of **F** "behave" quite differently in that respect. While the presence or absence of a DC or DC^{R} in a hammock is invariant under reduction, this is not the case for N and Q. For N, we find that a DN might be removed by reduction, in which case, however, a DC and and DC^{R} have to be present. For Q, a similar property holds, which, however, also involves Ψ and Ψ^{R}.

In the following lemma, we state these properties in a contrapositive manner. Therein, we consider a hammock H and its reduct G, and trace the fact that G has a minor in **F** back to the fact that H has a minor in **F** or a bare minor in **K**.

Lemma 7.2.1. *Let $H \Leftarrow G$, then an* **F***-minor of G implies an* **F***-minor or a bare* **K***-minor of H, as follows:*

1. *if $F \preccurlyeq G$, then $F \preccurlyeq H$ for $F \in \{\mathsf{C}, \mathsf{C}^{\mathrm{R}}\}$*

2. *if* $\mathsf{N} \preccurlyeq G$, *then* $(\mathsf{N} \preccurlyeq H \vee (\mathsf{C} \preccurlyeq H \wedge \mathsf{C}^{\mathrm{R}} \preccurlyeq H))$

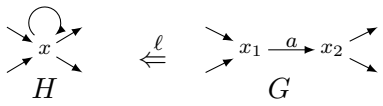
3. *if* $\mathsf{Q} \preccurlyeq G$, *then* $(\mathsf{Q} \preccurlyeq H \vee \mathsf{C} \preccurlyeq H \vee \mathsf{C}^{\mathrm{R}} \preccurlyeq H \vee \Psi \preccurlyeq_{\mathrm{b}} H \vee \Psi^{\mathrm{R}} \preccurlyeq H)$

Proof. We prove the claim separately for each reduction. For s- and p-reduction we show a stronger property for bulky graphs. The argument is downright trivial for these two reductions. For ℓ-reduction, however, we need to consider several subcases that lead to the details of the statement.

- $H \overset{s}{\Leftarrow} G$: Let $B \in \mathbf{B}$ be arbitrary and observe that H is a DG. Therefore, if G contains a DB, so does H. But since H is B-free by assumption, G is B-free, too.

- $H \overset{p}{\Leftarrow} G$: Let $B \in \mathbf{B}$ be arbitrary. Removing an arc from H does certainly not *introduce* any subgraph at all. This goes for a DB in particular.

- $H \overset{\ell}{\Leftarrow} H$: Let $l = xx$ denote the loop in H that allows for reduction, and let $a = x_1x_2$ be the constriction in G that results from splitting x. As a visual aid we sketch the relevant parts of H and G below:



  Suppose that G contains a minor F in $\mathbf{F}$. Let e realize $F \preccurlyeq G$ for $F = \mathsf{C}$ and suppose for the sake of contradiction that H is F-free. We distinguish by which of the x_i are pegs of F in G.

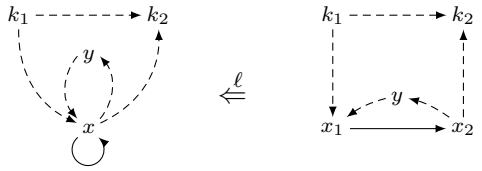
  If neither x_i is a peg, then each peg of F occurs in both G and H. It then follows from Prop. 6.1.1 that H contains a DF, too. Thus the assumption that H is F-free, is false.

  Next assume that exactly one of the x_i is a peg, say, x_1 (the argument is symmetric for x_2). We construct a map e' from V_F to V_G. This map is defined as e, except that the preimage q of x_1 is mapped to x in H. Now H contains an $(e'(q_1), e'(q_2))$-path for every q_1q_2-arc in F, where $q \neq q_i$, and these paths are pairwise internally disjoint. This follows from Prop. 6.1.1 and the fact the e is an embedding. Moreover, H contains an $(e'(q_1), x)$-path for each q_1q-arc of F, and an $(x, e'(q_2)$-path for each qq_1-arc. Again we find that these paths are internally disjoint. Therefore, e' is an embedding, i.e., H contains a DF. This contradicts F-freeness of H.

Finally, let both x_1 and x_2 be pegs of F in G, with preimages q_1 and q_2, respectively. Then Prop. 7.0.11 requires that $\mathrm{d}^+_F(q_1) \leq 1$ and $\mathrm{d}^-_F(q_2) \leq 1$ hold. We proceed by case distinction for F as in the claim.

1. Assume that $F = \mathsf{C}$: there is only one vertex q_2 that satisfies $\mathrm{d}^-_{\mathsf{C}}(q_2) \leq 1$ (cf. Fig. 7.1b). The vertex q_1 might be either vertex of the cycle of C, as each satisfies $\mathrm{d}^+_{\mathsf{C}}(q_1) \leq 1$. We fix one of these two vertices as q_1 and denote the other as y. Since C contains a q_1y-arc, G must contain an $(e(q_1), e(y))$-path. But the only out-arc of $e(q_1)$, which is a, is an in-arc of $e(q_2)$. In other words, $e(q_2)$ lies on every path from $e(q_1)$ to $e(y)$. Consequently, the paths in G that represent the arcs of C are not internally disjoint, which contradicts the assumption that e is an embedding. The same conclusion follows symmetrically for $F = \mathsf{C}^{\mathrm{R}}$.

2. For $F = \mathsf{Q}$, the degree restrictions on q_1 and q_2 are satisfied by two vertices each. The choices for the q_i lead to symmetric cases, due to the symmetry of Q. We find that Q contains a q_2q_1-arc, therefore G contains a path from x_2 to x_1. Observe that $H = G[x_1 = x_2]$, with merge vertex x. Thus the arcs that constitute the (x_2, x_1)-path in G form a cycle C in H, with x lying on C. Since ℓ-reduction is applicable in x, C is not guarded by x. In particular, this implies that C is not merely another x-loop, so C contains at least one more vertex y. The situation in G and H is sketched below; we denote a remaining pegs of Q in G, k_1 and k_2, these vertices are also present in H.



Since x does not guard C, there is at least one vertex on C that is not guarded by x either, i.e., not dominated or not co-dominated by x. We may wlog. assume that this vertex is y and that x does not dominate y; the case that x does not co-dominate y is symmetric. We apply Prop. 6.0.6 to x and y. Since x does not dominate y by assumption, this proposition implies that exactly on of the following holds in H: a) y dominates x, or b) some z dominates x and y, and H contains internally disjoint (z, x)- and (z, y)-paths.

   a) Assume y dominates x, then y also dominates k_1, since H contains a (k_1, x)-path that does not pass through y. So there is also a (y, k_1)-path

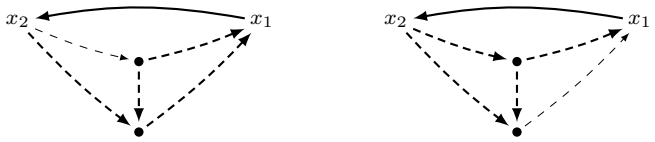
in H. Assume that at least one such path does not pass through k_2. In this case we find that C^{R} is a minor of H. The pegs of any corresponding DC^{R} are x, y, and k_2.

If, on the other hand, each (y, k_1)-path passes through k_2, then there is a (k_2, k_1)-path in H. We may assume that this path is internally disjoint with the (k_1, k_2)-path sketched above. Otherwise, we choose some k_1' that is an intersection vertex of the two paths and argue with k_1' instead of k_1. So k_1 and k_2 lie on a cycle. Moreover, y then also dominates k_2. Hence, there is a path from the source of H to y that does not intersect with any of the paths we are considering right now. Together, these paths form a $\mathrm{D}\Psi$ with the source of H being a peg. If this subdivision is bare, the claim follows. Otherwise, any bypass to the $\mathrm{D}\Psi$ implies the existence of a DC or a DC^{R}. An example showing the emergence of a $\mathrm{D}\Psi$, which also demonstrates that a bypass to that subdivision implies a DC or a DC^{R}, is given in Fig. 7.5.

b) Assume that there is some z with internally disjoint (z, x)- and (z, y)-paths in H. Since x and y lie on C, the paths from z enter C in distinct vertices. Thus follows $\mathsf{C} \preccurlyeq H$.

3. For $F = \mathsf{N}$, consider an appropriate DN in G. Since a is a constriction, this arc does not represent the trivial subdivision of any arc of N, as N is free of constrictions. By the same argument, it follows that a is not anti-parallel to an arc of N.

   Therefore, the preimages of x_1 and x_2 in N are non-adjacent, which is only satisfied by one pair of vertices of N, namely its source and sink (notice that N is a hammock). Moreover, the association of these vertices to x_1 and x_2 is uniquely determined by the in- and out-degrees following Prop. 7.0.11: the source of N maps to x_2, and the sink of N maps to x_1. Consequently, G contains a DN, together with a "back arc" from the sink to source of this DN. Such a graph contains both a DC and a DC^{R}, which becomes clear from the sketch below, where the subdivision of C (left), resp. C^{R} (right), is drawn bold. An example showing the emergence of a DN upon ℓ-reduction is given in Fig. 7.6.



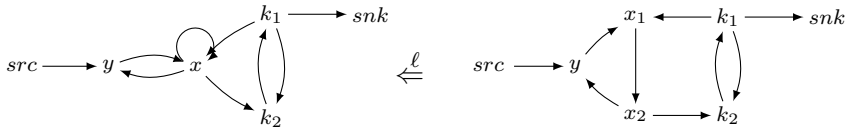


Figure 7.5: Example for ℓ-reduction producing a DQ. Observe that either side contains a bare $\mathsf{D}\Psi$.

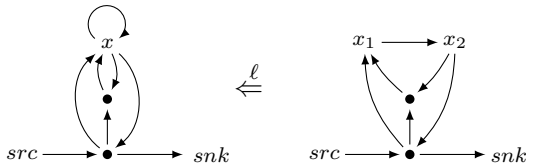


Figure 7.6: Example for ℓ-reduction producing a DN. Observe that either side contains both a DC and a DC^{R}.

So we have found $\mathsf{C} \preccurlyeq G$ and $\mathsf{C}^{\mathrm{R}} \preccurlyeq G$ in this last case. As we have already shown, $\mathsf{C} \preccurlyeq H$ and $\mathsf{C}^{\mathrm{R}} \preccurlyeq H$ hold as well in that case.

For the reductions $H \overset{s}{\Leftarrow} G$ and $H \overset{p}{\Leftarrow} G$ we have shown that $F \preccurlyeq G$ implies $F \preccurlyeq G$ for $F \in \mathbf{F}$. Considering the reduction $H \overset{\ell}{\Leftarrow} G$, a $\mathbf{F}$-minor of G implies an $\mathbf{F}$-minor of H or a bare $\mathbf{K}$-minor of H as broken down in the claim. Thus the claim holds and the proof is concluded. □

Next we consider the effects of reduction on hammocks that do have a minor in $\mathbf{F}$. In this case, for a change, we can make use of properties that were already established. We find that minors, resp. subdivisions, from $\mathbf{F}$ are not removed upon reduction.

Lemma 7.2.2. *Let $H \Leftarrow G$, then $F \preccurlyeq H$ implies $F \preccurlyeq G$ for $F \in \mathbf{F}$.*

Proof. We know that a reduction allows for the complementary expansion, i.e., $H \Leftarrow G$ implies $G \Rightarrow H$. Assume now that G is $\mathbf{F}$-free and reconsider Lem. 7.1.1, resp. its proof: therein we have shown that $B \preccurlyeq G$ implies $B \preccurlyeq H$ for bulky B and $G \overset{s}{\Rightarrow} H$ or $G \overset{p}{\Rightarrow} H$. Since $\mathbf{F} \subseteq \mathbf{B}$, this holds for $B \in \mathbf{F}$ as well. For $G \overset{\ell}{\Rightarrow} H$, we have shown that $B \preccurlyeq G$ yields $B' \preccurlyeq H$, for bulky B, B', and indirectly, that $\mathrm{d}_B^-(v) < 2$ or $\mathrm{d}_B^+(v) < 2$ for some $v \in V_B$ implies $B = B'$. This degree property is satisfied by each $F \in \mathbf{F}$, so $F \preccurlyeq G$ implies $F \preccurlyeq H$ for ℓ-reduction, too.

Thus, if $H \Leftarrow G$, the assumption that H contains a minor in $\mathbf{F}$, while G is $\mathbf{F}$-free, is contradictory. Equivalently, the claim follows. □

As for minors, resp. their subdivisions, in $\mathbf{F}$ before, we study the effects of spl-reductions on minors in $\mathbf{K}$. Again, we first investigate if, or rather "how", such minors can be the result of a reduction.

Lemma 7.2.3. *Assume $H \Leftarrow G$ and let $K \in \mathbf{K}$ be fixed. Then $K \preccurlyeq_{\mathrm{b}} G$ implies $K \preccurlyeq_{\mathrm{b}} H \vee \mathsf{C} \preccurlyeq H \vee \mathsf{C}^{\mathrm{R}} \preccurlyeq H$.*

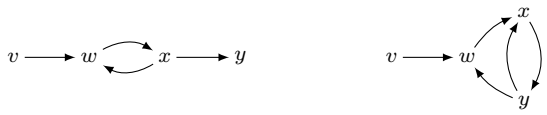
Proof. Observe that each $K \in \mathbf{K}$ is free of parallel arcs. If all pegs of K in G also occur in H, i.e., if $e_V(V_K) \subseteq V_H \cap V_G$ holds, then Prop. 6.1.1 states that the connectivity among these pegs is invariant under reduction. This is true for paths representing arcs of K, as well as for bypasses. In other words, bypasses are not removed upon reduction if all pegs occur in both H and G, i.e., a nonbare embedding does not "become" bare in this case. It is thus sufficient to consider cases where pegs are introduced with $G \Leftarrow H$. Notice that the removal of a peg is irrelevant to this proof.

Since $V_H = V_G$ holds for $H \overset{p}{\Leftarrow} G$, nothing needs to be done for this case, following the discussion above. Regarding $H \overset{s}{\Leftarrow} G$, a peg is removed at most, so the statement follows for this case, too.

It remains to consider the reduction $H \overset{\ell}{\Leftarrow} G$, whereupon the number of vertices increases. Let $l = qq$ be the loop in H that allows for reduction, and let $a = q_1q_2$ denote the constriction that emerges from reduction. Assume that $K \preccurlyeq_{\mathrm{b}} G$ is realized by e. We consider the cases where at least one of the q_i is a peg of K; otherwise, the claim follows immediately.

Let q_1, be the peg of $k \in V_K$ and let q_2 be no peg. Since $\mathrm{d}^+_G(q_1) = 1$, Prop. 7.0.11 implies that at most one arc leaves k in K. Then $K \preccurlyeq H$ is realized by e', which is as e, except that $e'(k) = q$, the in-arcs of k in K are mapped to in-paths of q and the sole out-arc, if it exists, is mapped to an out-path of q. Since e is a bare embedding, no bypass possibly intersects the out-path of q_1 by assumption, in particular not in q_2. Hence no bypass is accidentally "removed" by using e' as an embedding, so e' realizes $K \preccurlyeq_{\mathrm{b}} H$, which satisfies the claim.

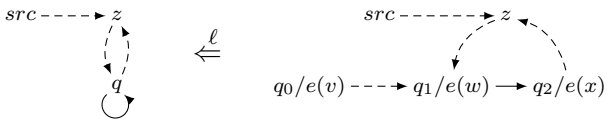
If q_1 and q_2 are both pegs wrt. e, some additional effort is required. In this case, we treat Φ and Ψ explicitly; for Ψ^{R}, the claim follows from that for Ψ by directional duality. As in the proof of Lem. 7.1.5, the vertices of Φ and Ψ are denoted as shown below:



1. We start with $K = \Phi$. Since v and y are not adjacent in Φ, the case where the q_i are pegs of these two vertices immediately yields a as a bypass in G. As a bare embedding, e maps at least one of w and x to a q_i. We assume wlog. $q_1 = e(w)$ and examine the preimage of q_2.

   If q_2 is the peg of v or y, it follows immediately that e is not a bare embedding, since a constitutes a bypass.

   So suppose $q_2 = e(x)$, then G contains an (q_2, q_1)-path, or equivalently, an $(e(x), e(w))$-pat P. We find that P contains more than one arc: otherwise, ℓ-reduction is not applicable to l in H, since q guards an arc that is parallel to l. So let z be an internal vertex of P, then z is also present in H; more specifically, H then contains a (q, z)- and a (z, q)-path. Since ℓ-reduction is applicable to l, the vertex q does not guard z in H. This means that H contains a (src, z)-path or a (z, snk)-path that does not pass through q. We proceed with the first possibility and let P_z this path. The part of H and G that we are considering looks as follows:

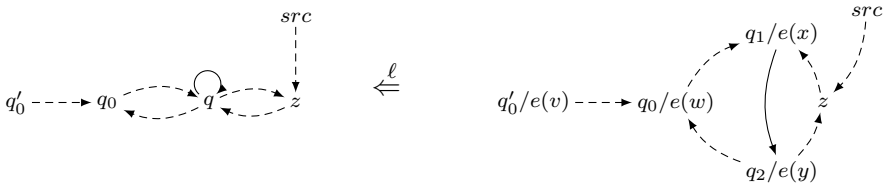


   Consider any path P_{q_0} from the source of G to q_0, the peg of v. Notice that if $q_0 = src$ holds in G, then a bare embedding e' that realizes $\Phi \preccurlyeq_\mathrm{b} H$ is given by $e'(w) = z$ and $e'(x) = q$ and letting e' otherwise be identical to e. In that case, the claim follows, so assume $q_0 \neq src$, which holds for both G and H. If P_{q_0} passes through neither q_1 nor q_2, we find $\mathsf{C} \preccurlyeq G$ and, following Lem. 7.2.1, also $\mathsf{C} \preccurlyeq H$. On the other hand, if P passes through q_1 or q_2, then G contains a bypass to the uncovered $\mathrm{D}\Phi$. This contradicts the assumption that e is a bare embedding. If we choose P_z as a (z, snk)-path that does not pass through q, and P_{q_0} as a (q_0, snk)-path, a symmetric argument yields either $\mathsf{C}^\mathrm{R} \preccurlyeq H$ or that e is not bare. This concludes the argument for Φ.

2. Now let $K = \Psi$. Since e is bare by assumption, we find $q_2 \neq e(v)$ immediately, because v has no in-arc in Ψ, yet q_1 is a peg, too. If we assume $q_1 = e(v)$, we construct an embedding e' that realizes $\Phi \preccurlyeq_\mathrm{b} H$: e' maps the preimage of q_2 to q and v to any predecessor of q; otherwise e' is as e. Since v has no

in-arcs and e is bare, e' is bare too. This covers the case that either q_i is the peg of v, and we consider the remaining cases.

Let us first assume $q_1 = e(x)$ and $q_2 = e(y)$. This case is akin to the nontrivial case for Φ, which we discussed at length above. Similarly, we find that G contains a (q_2, q_1)-path, which, due to ℓ-reduction being applicable to H, consists of at least two arcs. Again, let z denote a vertex on this path and notice that q does not guard z in H. As before, assume wlog. that H contains a path from its source to z that does not pass through q. The relevant parts of H and G are shown below:



Now let $e(v) = q_0'$ and consider a (src, q_0')-path P in G. If none of q_0, q_1, or q_2 lies on P, then $\mathsf{C} \preccurlyeq G$ follows, which further yields $\mathsf{C} \preccurlyeq H$. Otherwise, a segment of P constitutes a bypass to $e(\Psi)$, contrary to the assumption that e is bare. Again, if we consider paths to snk in G, instead of paths from src, we find $\mathsf{C}^{\mathrm{R}} \preccurlyeq H$ symmetrically.

The final case for Ψ is that one q_i is the peg of w while the other is the peg of either x or y. The arguments are symmetric if x and y are interchanged; we pursue the argument for x. First, let $q_1 = e(w)$ and $q_2 = e(x)$. Since we have $\mathrm{d}^-_\Psi(x) = 2$, Prop. 7.0.11 implies $\mathrm{d}^-_G(q_2) \geq 2$. This, however, is infeasible with ℓ-reduction, which produces a constriction, i.e., $\mathrm{d}^-_G(q_2) = 1$. On the other hand, if we assume $q_1 = e(x)$ and $q_2 = e(w)$, then the q_1q_2-arc already represents an $(e(x), e(w))$-bypass, so e is not bare.

This concludes the proof. □

Lemma 7.2.4. *Let $H \Leftarrow G$, then $K \preccurlyeq_{\mathrm{b}} H$ implies $K \preccurlyeq_{\mathrm{b}} G$ or $F \preccurlyeq G$ for $K \in \mathbf{K}$, $F \in \mathbf{F}$, as follows:*

- *if $\Phi \preccurlyeq_{\mathrm{b}} H$, then $(\Phi \preccurlyeq_{\mathrm{b}} G \vee \mathsf{C} \preccurlyeq G \vee \mathsf{Q} \preccurlyeq G)$*
- *if $\Psi \preccurlyeq_{\mathrm{b}} H$, then $(\Phi \preccurlyeq_{\mathrm{b}} G \vee \mathsf{C} \preccurlyeq G \vee \mathsf{N} \preccurlyeq G)$*

Proof. As in Lem. 7.2.3, the statement follows immediately from Prop. 6.1.1 if all pegs of K occur in $V_H \cap V_G$. In particular, nothing needs to be done for

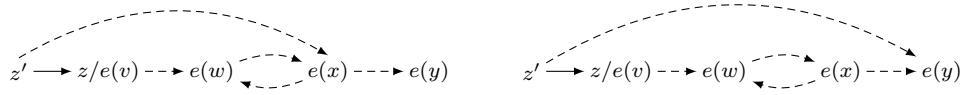
$H \overset{p}{\Leftarrow} G$. In the following, assume that $K \preccurlyeq_{\mathrm{b}} F$ is realized by e for $K \in \mathbf{K}$. As usual, we only consider Φ and Ψ.

For $H \overset{\ell}{\Leftarrow} G$ let $l = xx$ be the loop that is reduced in H and let $a = x_1x_2$ denote the constriction introduced in G. Observe that each $k \in V_K$ satisfies $\mathrm{d}_K^-(k) \leq 1$ or $\mathrm{d}_K^+(k) \leq 1$. Thus, if $x = e(k)$, we can always define an embedding e' of K in G that maps k to x_1 or x_2 depending on the degrees of k. The connectivity among the pegs of K in G then is the same as in H, thus e' realizes $K \preccurlyeq_{\mathrm{b}} G$.

In the case $H \overset{s}{\Leftarrow} G$, let z denote the simple vertex of H that allows for reduction. Assume that z is a peg of K. Since z is simple in H, Prop. 7.0.11 implies that in K, the preimage of z satisfies $\mathrm{d}_K^-(e^{-1}(z)) \leq 1$ and $\mathrm{d}_K^+(e^{-1}(z)) \leq 1$.

There are two vertices with this property in Φ and one in Ψ. We choose v as the preimage of z for either graph. Observe that the in-degree of v is zero. Since z is simple, H contains a unique predecessor z' of z. Moreover, since e is a bare embedding, H contains no path from any other peg to z. This implies that z' is not also a peg of K wrt. e, and that there is no path from any other peg of K to z'. The embedding e' of K into H is now defined by mapping v to z' and being otherwise as e. If e' is bare, the claim follows. Otherwise, we observe that no bypass wrt. e enters z, hence no bypass wrt. e' enters z'. So if e' is not bare, a bypass must leave $e'(v) = z'$; the connectivity among all other pegs does not differ in e and e'.

There are two possibilities for a bypass like that for $K = \Phi$ in H, they are sketched below:



In the case on the left, we find a DC, while on the right, we find a DQ. Thus follows the claim for $K = \Phi$. For $F = \Psi$, the possibilities in H are as follows:

In either case, we find a DC, while in the case on the right, we also find a DN. These proves the claim for Ψ, and the statement for Ψ^{R} follows by symmetry. □

We have thus found that $\mathbf{F} \cup \mathbf{K}$ behaves stable wrt. being (bare) minors in hammocks under spl-operations. Although such minors are exchanged for another in certain cases, they never appear by manipulating **F**- and **K**-free minors; likewise, they never disappear altogether. By omitting the finer details of Lems. 7.2.1, 7.2.2, 7.2.3 and 7.2.4, we arrive at a cornerstone result regarding the characterization of **SPL** by forbidden minors.

Theorem 7.2.5. *Let $H_1, H_2 \in \mathbf{H}$ s.t. $H_1 \Rightarrow H_2$ or $H_1 \Leftarrow H_2$. Then H_1 is free of **F**-minors and bare **K**-minors iff H_2 is.*

Most importantly, we can associate (bare) minors in an arbitrary hammock with (bare) minors in its unique normal form under spl-reductions.

Corollary 7.2.6. *Let $H \in \mathbf{H}$, then H is free of **F**-minors and bare **K**-minors iff $\mathrm{R}(H)$ is.*

It remains to show that **F** and **K** are sufficient to characterize **SPL**. Since we can restrict further investigation to spl-normal hammocks, this comes down to showing that an spl-normal hammock that is not the axiom graph P_1 has a (bare) minor in **F** or **K**. To this end we introduce an auxiliary class of graphs.

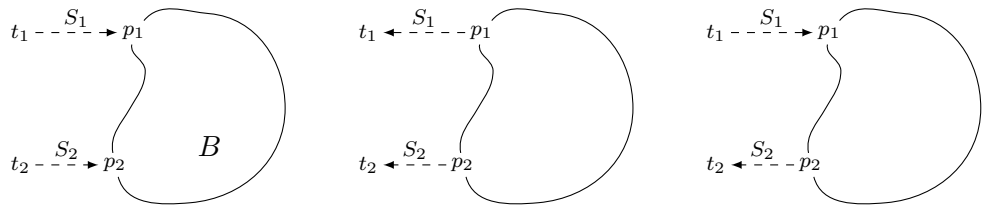
Definition 15. A *kebab* is a graph consisting of three arc-disjoint subgraphs: a strong component B, called the *body*, and two nonempty, disjoint paths S_1 and S_2, called the *spikes* of the kebab. Exactly one endpoint of either spike is a vertex of B.

Some additional terminology proves useful for dealing with kebabs. The endpoint of a spike that connects the spike to the body is the *puncture* of this spike, the opposite endpoint is its *tip*. A spike that enters the body of a kebab is an *in-spike*, one that leaves the body is an *out-spike*.

Kebabs are further differentiated wrt. the orientation of their spikes. Let K be a kebab with spikes S_1 and S_2. If both S_i are in-spikes, K will be called an *in*-kebab; if both S_i are out-spikes, we call K an *out*-kebab. If, say, S_1 is an in-spike while S_2 is an out-spike, K is referred to as an *inout*-kebab. The three types are sketched in Fig. 7.7.

In order to prove two essential lemmas about kebabs, we need some technical preliminaries concerning hammocks and strong graphs.

Proposition 7.2.7. *Let (H, src, snk), $H \neq \mathsf{P}_1$, be spl-normal and let v be a vertex of H that does not carry a loop. Then v is incident to at least three nonparallel proper arcs.*



(a) in-kebab (b) out-kebab (c) inout-kebab

Figure 7.7: Possible structures of a kebab with body B, spikes S_i, and respective tips t_i and punctures p_i for $i \in \{1, 2\}$

Proof. Let H and v be as in the claim. First we observe that since H is p-normal, any pair of arcs in H is nonparallel. Since H is a hammock, it contains a shortest (src, v)-path and a shortest (v, snk)-path. Let $a_1 = uv$ and $a_2 = vw$ be the in- and the out-arc of v on these paths. Obviously, a_1 and a_2 are distinct and proper. If v does not carry a loop, then v must be incident to a third arc a_3, since otherwise H could be s-reduced. □

Proposition 7.2.8. *Let G be a strong graph with distinct vertices x and y. Then there is a cycle $C \subseteq G$ with distinct vertices $z_x, z_y \in V_C$, s.t. G contains an (x, z_x)-path and a (y, z_y)-path which are disjoint.*

Proof. Since G is strong, it contains paths among x and y in either direction. Let P_1 denote a shortest xy-path and P_2 a shortest yx-path in G. The set of crossing vertices Z is defined as the set of internal vertices shared by the P_i, formally $Z := (V_{P_1} \cap V_{P_2}) \setminus \{x, y\}$.

- If $Z = \emptyset$, then P_1 and P_2 form a cycle already, and we choose $z_x = x$ and $z_y = y$; in this case, P_x and P_y are both empty.
- If $Z = \{z\}$, the claim follows for $z_x = x$ and $z_y = z$, or $z_x = x$ and $z_y = z$; here, either P_x or P_y is empty, but not both (Fig. 7.8a).
- Finally if $|Z| \geq 2$, let z_x and z_y be consecutive crossing vertices, i.e., assume that no $z \in Z$ lies between z_x and z_y on P_1, resp. P_2 (Fig. 7.8b). These vertices then satisfy the claim.

□

With these preparations we are set to prove that a hammock that is not contained in **SPL** necessarily has a minor in **F**, or a bare minor in **K**. The

(a) $|Z| = 1$ (b) $|Z| \geq 2$

Figure 7.8: Crossing vertices of an (x, y)-path and a (y, x)-path; cases in the proof of Prop. 7.2.8.

work that is involved in the proof is divided over the following two lemmas, and the pieces are put together in Thm. 7.2.11. First, we show that the existence of a kebab is sufficient to imply the existence of a forbidden minor in a reduced hammock.

Lemma 7.2.9. *Let $G \in \mathbf{H}$ be spl-normal and assume that G contains a kebab. Then $F \preccurlyeq G$ for some $F \in \mathbf{F}$ or $\Phi \preccurlyeq_{\mathrm{b}} G$.*

Proof. Let (G, src, snk) be an spl-normal hammock s.t. G contains at least one kebab. Among the kebabs in G, we choose K of maximal size, i.e., no kebab $K' \subseteq G$ contains more arcs than K. Let S_1 and S_2 denote the spikes, and B the body of K. The cases for K being an in-kebab or an out-kebab are symmetric, so the two main cases of this proof distinguish whether K is an in-kebab or an inout-kebab. Therefore, one spike of K is definitely an in-spike, say, S_1.

Other than that, let t_i denote the tip and p_i the puncture of the spike S_i throughout the proof.

Let K be an in-kebab,

and apply Prop. 6.0.6 to the tips of K. According to that proposition, exactly one of the following holds in G:

a) t_1 dominates t_2,

b) t_2 dominates t_1, or

c) some $x \in V_G$ dominates either t_i, and G contains internally disjoint (x, t_1)- and (x, t_2)-paths.

We proceed with case distinction according to theses properties. As the cases a) and b) are symmetric, we elaborate on b) and c).

b) Assume that t_2 dominates t_1 and let P be a shortest (t_2, t_1)-path in G. If P and B are disjoint, then P contains an (S_2, S_1)-segment. Using Prop. 7.2.8 we find $\mathsf{C} \preccurlyeq G$ (Fig. 7.9a).

So let P and B intersect, then P contains a unique "terminal" segment P': the (B, t_1)-subpath that is internally disjoint with B. We show that our choice of K implies that P' consists of a single arc. First, P' is also internally disjoint with S_1: otherwise we could add arcs from P' and S_1 to B to find a kebab whose body is bigger than B. Second, if P' consists of several arcs, we can remove its arc that leads out of B and "use" the remaining segment to extend S_1 to a longer spike. Either assumption leads to a contradiction with our choice of K. It follows that P' consists of a single arc $a = bt_1$ for some $b \in V_B$ (Fig. 7.9b).

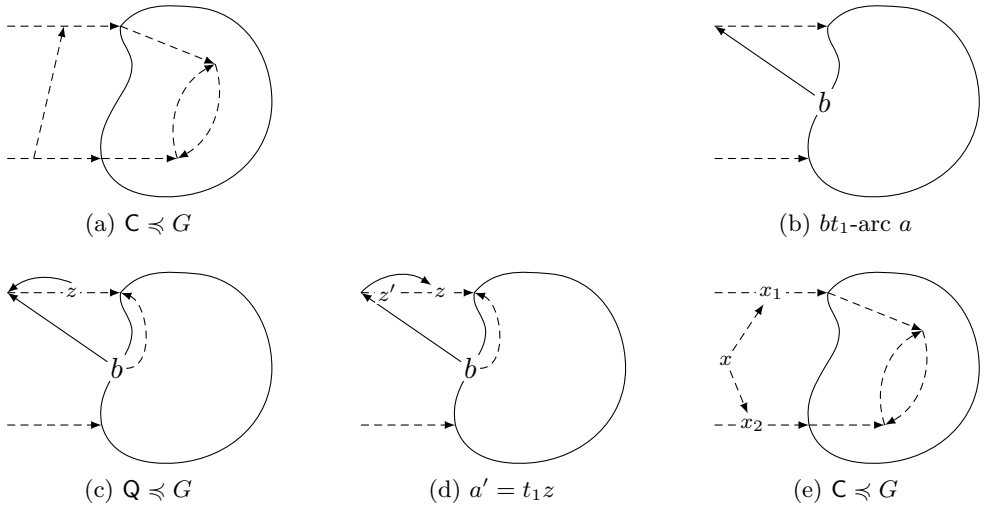
We infer by contradiction that t_1 is incident to a third proper arc. Suppose this claim to be false. Then, since G is s-reduced, t_1 carries a loop. Since G also is ℓ-reduced, t_1 guards vertex x. Now t_1 is part of a strong subgraph B', which consists of S_1, B, and P', where $V_{B'} = V_{S_1} \cup V_B$ (cf. Fig. 7.9b). Hence if $x \in V_{B'}$, then t_1 guard all of B' and must be incident to an arc of a (src, t_1)- or (t_1, snk)-path. If $x \notin V_{B'}$, then t_1 is incident to an arc of the (t_1, x)-path. In either case, we find a third incident proper arc, as claimed.

Let this arc be $a' = t_1 z$ or $a' = z t_1$. Our choice of K implies $z \in V_K$: notice that B', the strong subgraph we just found, properly contains B. If z lies outside K, we find a kebab with body B', in-spike S_2, and the arc a', interpreted as a path, as a second in- or out-spike. This contradicts the property that the body of K is arc-maximal among all kebabs in G.

It remains to consider the location of z in K for this case. With the help of Prop. 7.2.8, we see (from Fig. 7.9b) that $z \in V_{S_2}$ yields $\mathsf{C} \preccurlyeq G$, and that $z \in V_B$ yields $\mathsf{C} \preccurlyeq G$ or $\mathsf{C}^{\mathrm{R}} \preccurlyeq G$ (depending on a's orientation).

So assume $z \in V_{S_1} \setminus \{p_1\}$ and consider the orientation of a'. If $a' = z t_1$ we find $\mathsf{Q} \preccurlyeq G$, with pegs t_1, p_1, b, and z (Fig. 7.9c). On the other hand, $a' = t_1 z$ leads to a contradiction: Since G is p-normal, at least one vertex z' lies between t_1 and z on S_1; omitting the $t_1 z'$-segment of S_1 produces an in-kebab with tips z' and t_2 and a body properly containing B (Fig. 7.9d). Since the body K is arc-maximal among all kebabs in G this can not be the case.

b) Let $x \in V_G$ be such that G contains a (x, t_1)-path P_1 and a (x, t_2)-path P_2, which are internally disjoint. If neither P_i intersects B, we find $\mathsf{C} \preccurlyeq G$ with



(a) $\mathsf{C} \preccurlyeq G$ (b) bt_1-arc a (c) $\mathsf{Q} \preccurlyeq G$ (d) $a' = t_1 z$ (e) $\mathsf{C} \preccurlyeq G$

Figure 7.9: Cases occurring in the proof of Lem. 7.2.9 if G contains an in-kebab. Solid arrows represent arcs, dashed arrows represent paths.

help of Prop. 7.2.8; this is shown in Fig. 7.9e, where x_i denotes the "first" vertex on P_i that is also in S_i.

If, say, P_1 intersects B, let b be the last vertex on P_1 that is in B and let x be the first vertex on P_1 that is in $V_{S_i} \setminus \{p_i\}$. If $x \neq t_1$, we find a kebab in G with a body containing B, contradicting our choice of K. For $x = t_1$, the claim was proven in the previous case already (cf. Fig. 7.9b).

Let K be an inout-kebab,

where, wlog., S_1 is the in-spike and S_2 the out-spike. From Prop. 7.2.8 follows $\Phi \preccurlyeq_\mathrm{b} K$, where the tips of K are pegs of Φ wrt. some embedding e. This is shown in Fig. 7.10a. If e also realizes $\Phi \preccurlyeq_\mathrm{b} G$, i.e., if G contains no bypass to $e(\Phi)$, the claim follows already.

Assume that e is not bare, then G contains a bypass P. It follows from the structure of Φ that one of the tips of K is an endpoint of this bypass. Choosing between t_1 and t_2 leads to symmetric cases; we proceed with t_1. Then P is either a (t_1, q)-path or a (q, t_1)-path, where q is a further peg of Φ wrt. e.

- We first consider the case where P is a (t_1, q)-path and distinguish by the preimage of q in Φ under e. Considering the structure of Φ, this requires to

investigate the cases $q = e(x)$ and $q = e(y)$ (cf. Fig. 7.10a).

In the first case, $q = e(x)$, it follows that P is a $(t_1, e(x))$-path that deviates at some point from the path with segments S_1 and the $(p_1, e(x))$-path in B (as drawn in Fig. 7.10a). Following Prop. 7.2.8, we find $\mathsf{C} \preccurlyeq G$ (Fig. 7.10b).

For $q = e(y)$ we find further subcases that lead to different minors in $\mathbf{F}$. We consider the cycle C of the $\mathrm{D}\Phi$ in G (cf. Fig. 7.10a), and ask whether P and C are disjoint.

If so, we find $\mathsf{Q} \preccurlyeq G$ immediately (Fig. 7.10c), so assume P and C intersect. Since P is a bypass of the found $\mathrm{D}\Phi$, at least one of $e(w)$, $e(x)$ does not lie on P. If P does not pass through $e(w)$, then an initial segment of P is a (t_1, C)-path, which yields $\mathsf{C} \preccurlyeq G$, similar to the case shown in Fig. 7.10b. If P does not pass through $e(x)$, we find $\mathsf{C}^{\mathrm{R}} \preccurlyeq G$ by taking the terminal (C, t_2)-segment of P into account (Fig. 7.10d).

- Next, let P be a (q, t_1)-path and let k denote the predecessor of t_1 on P. Our choice of K requires $k \in V_K$: otherwise, S_1 could be augmented to a longer spike, which contradicts our choice of K. We distinguish by the location of k in K.

  For $k \in V_B$, the argument is identical to a case we already considered for K being an in-kebab (replace b with k in Figs. 7.9b to 7.9d). The difference is that this time, S_2 is an out-spike, which, however, is irrelevant. From this previous part of our proof follows $\mathsf{Q} \preccurlyeq G$.

  Next, suppose that k is an internal vertex of S_2, i.e., $k \in V_{S_2} \setminus \{p_2, t_2\}$. Observe that B, the (p_2, k)-segment of P, the kt_1-arc, and S_1 form a strong graph $B' \subseteq G$ that properly contains B. Moreover, we find that the source src of G is no vertex of K: this follows because $\mathrm{d}^-(src) = 0$ by definition, yet $\mathrm{d}^-(q) \geq 1$ for all $q \in V_K$. But then, there is a nonempty (s, B')-path in G. This path and the nonempty (k, t_2)-segment of S_2 form a pair of spikes to the body B'. The resulting kebab contradicts our choice of K, hence k can not be an internal vertex of S_2.

  Now if $k = t_2$, then the bypass consists merely of a t_2t_1-arc. In that case we invoke Prop. 7.2.7 to find that either t_i is incident to a third arc a_i. Let z_i denote the vertex adjacent to t_i via a_i. Since K and the bypass P form a strong graph that properly contains B, at least one z_i is a vertex of K: otherwise, once more we find a kebab with bigger body than K. Using symmetry again, it is sufficient to treat $z_1 \in V_K$. We need to distinguish by the orientation of a_1.

– First, we assume that z_1 is a predecessor of t_1, meaning that $a_1 = z_1 t_1$. We further distinguish by the location of z_1 in K, i.e., whether z_1 is a vertex of either spike or the body.

  If $z_1 \in V_{S_1}$, the fact that a_1 is a proper arc implies $z_1 \neq t_1$. Now if z_1 is the puncture of S_1, $z_1 = p_1$, we find $\mathsf{C} \preccurlyeq G$, since B contains a (p_2, p_1)-path (Fig. 7.10e). On the other hand, if z_1 is an internal vertex of S_1, we find $\mathsf{Q} \preccurlyeq G$ with pegs t_1, z_1, p_1, and p_2 (Fig. 7.10f).

  Next, suppose $z_1 \in V_{S_2}$; this time, $z_1 \neq t_2$ follows, since G is p-normal and the bypass P constitutes of a $t_2 t_1$-arc already. For any other $z_1 \in V_{S_2} \setminus t_2$, we observe that S_1, B, and the (p_2, z_1)-segment of S_2 form a strong graph B' that properly contains B. Certainly, the source src of G is not contained in B' since $\mathrm{d}^-(src) = 0$, so G contains a shortest nonempty (src, B')-path P'. Consider whether z_1 lies on P'. If z_1 does not lie on P', G contains an inout-kebab with body B', in-spike P' and the (z_1, t_2)-segment of S_2 as the out-spike. If otherwise z_1 lies on P', we find an in-kebab with body B', and in-spikes P' and P. Either case contradicts our choice of K, hence $z_1 \notin V_{S_2}$ follows.

  Let finally $z_1 \in V_B$. We may actually assume $z_1 \in V_B \setminus \{p_1, p_2\}$ as the cases for z_1 being a puncture have been dealt with already. Following Prop. 7.2.8, we find $\mathsf{C}^{\mathrm{R}} \preccurlyeq G$, once again with help of Prop. 7.2.8 (Fig. 7.10g).

– Now let z_1 be a successor of t_1, adjacent by $a_1 = t_1 z_1$. As in the previous case, we consider the possible locations of z_1 in K separately.

  For $z_1 \in V_{S_1}$, again we first notice that since a_1 is a proper arc, $z_1 \neq t_1$ holds. Since G is p-normal, some vertex k_1 lies between t_1 and z_1 on S_1 (Fig. 7.10h). We have to proceed with considering the arc a_2, which is incident to t_2. As for a_1, we distinguish by the orientation of a_2 and the location of z_2 in K. We skim over most of the cases rather quickly, the claimed minors or bigger kebabs are found by adding the particular a_2 to Fig. 7.10h.

  If z_2 lies outside K, we delete either the (t_1, k_1)-segment of S_1 to find a kebab with a_2 and the (k_1, z_1)-segment of S_1 as spikes and a body that properly contains B. If z_2 is a puncture of K, we either find a kebab with bigger body than K or one of Q, C, or C^{R} as a minor, depending on the orientation of a_2 and the p_i. If z_1 lies in B, we find a bigger kebab or Q as a minor. For $z_2 \in V_{S_2} \setminus p_2$ where $a_2 = t_2 z_2$, we get $\mathsf{Q} \preccurlyeq G$ (cf. Fig. 7.10f, which shows the symmetric case for a_1). If $a_2 = z_2 t_2$, we know

that some k_2 lies between z_2 and t_2 on S_2, for G is p-normal. This yields a bigger kebab by removing a segment from each S_i (Fig. 7.10i). For $z_2 \in V_{S_1} \setminus p_1$ where $a_2 = z_2 t_2$ we find $\mathsf{Q} \preccurlyeq G$. Finally, for $z_2 \in V_{S_1} \setminus p_1$ where $a_2 = t_2 z_2$ we find a bigger kebab if z_2 lies between z_1 and p_1, and $\mathsf{N} \preccurlyeq G$ is z_2 lies between t_1 and k_2.

Next, for $z_1 \in V_{S_2}$ two possibilities arise: for $z_1 = p_2$ we use Prop. 7.2.8 once more to find $\mathsf{C} \preccurlyeq G$. If otherwise $z_1 \in V_{S_2} \setminus p_2$, we find $\mathsf{Q} \preccurlyeq G$.

Finally, for $z_1 \in V_B$, we may again assume $z_1 \in V_B \setminus \{p_1, p_2\}$ as the p_i were already considered in the cases where $z_1 \in V_{S_i}$. With $z_1 \neq p_1$ we find $\mathsf{C} \preccurlyeq G$ with help of Prop. 7.2.8. This case is largely identical to the previous one where $z_1 = p_2$.

This exhausts all possible cases and the proof is complete.

□

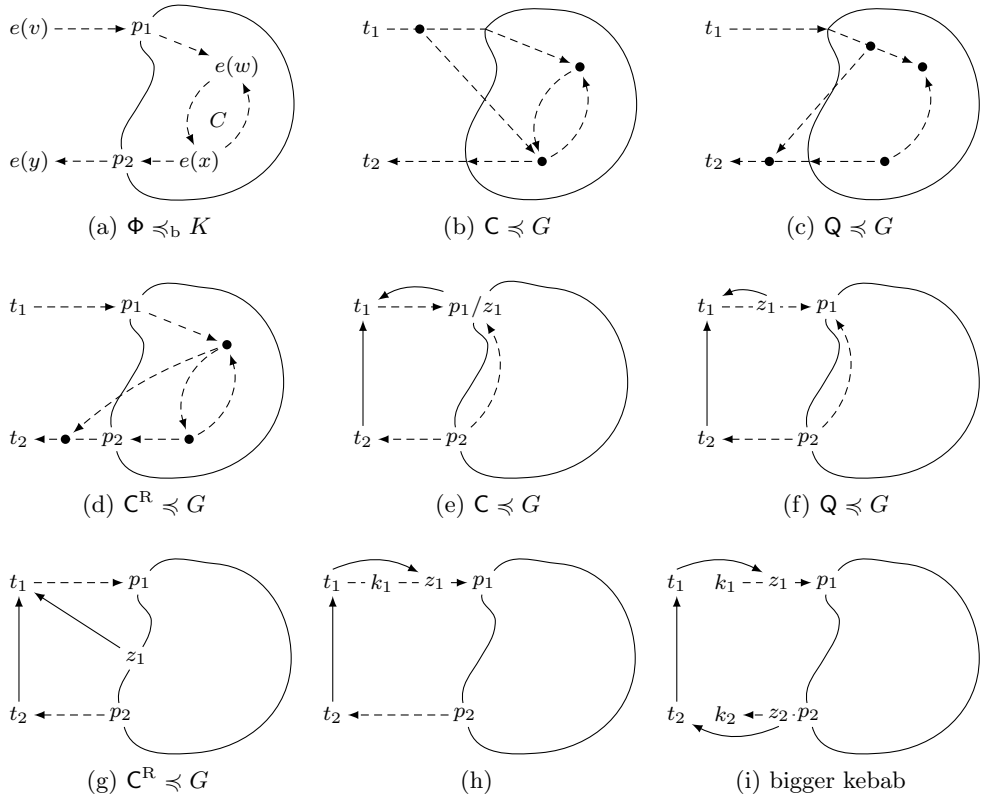
Lemma 7.2.10. *Let G be an spl-normal hammock with cycles. Then $F \preccurlyeq G$ for some $F \in \mathbf{F}$ or $F \preccurlyeq_{\mathrm{b}} G$ for some $F \in \mathbf{K}$.*

Proof. Let G be as in the claim, thus, in particular, assume that G is not acyclic. Let *src* and *snk* denote the source and the sink of G, respectively.

We first show that G contains a proper cycle, i.e., one that is not merely a loop. This is seen as follows: if G is free of loops, it necessarily contains at least one proper cycle since G is not acyclic by assumption. Otherwise, let $a = xx$ be any loop of G. Since G is ℓ-normal by assumption, x guards a further arc $a' = yz$. Now if a' is a loop, too, we find $x = y = z$. Then, however, a and a' are both x-loops, which can not be, as G is p-normal. So G contains an (x, y)- and a (z, x)-path, which form a proper cycle.

We choose a smallest proper cycle $C \subseteq G$. Since C is proper, it contains at least two distinct vertices. Consider the source and sink of G. Neither of these vertices lies on C, since every vertex of C has in- and out-degree at least one. So G contains a shortest (src, C)-path P_{src} and a shortest (C, snk)-path P_{snk}. Let x denotes the vertex of C where P_{src} enters C, and let y denote the vertex where P_{snk} leaves C. We distinguish whether x and y coincide.

1. If $x \neq y$, we further distinguish whether P_{src} and P_{snk} are disjoint; the two cases are sketched below.

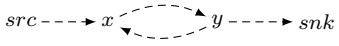


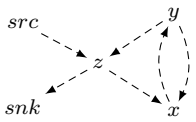
(a) $\Phi \preccurlyeq_{\mathrm{b}} K$ (b) $\mathsf{C} \preccurlyeq G$ (c) $\mathsf{Q} \preccurlyeq G$

(d) $\mathsf{C}^{\mathrm{R}} \preccurlyeq G$ (e) $\mathsf{C} \preccurlyeq G$ (f) $\mathsf{Q} \preccurlyeq G$

(g) $\mathsf{C}^{\mathrm{R}} \preccurlyeq G$ (h) (i) bigger kebab

Figure 7.10: Cases occurring in the proof of Lem. 7.2.9, where it is assumed that G contains an inout-kebab.





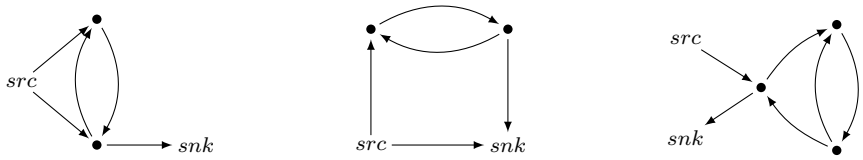
If P_{src} and P_{snk} are disjoint, we have found an inout-kebab with body C, in-spike P_{src}, and out-spike P_{snk}. This can be seen above on the left. In that case, the claim follows by virtue of Lem. 7.2.9.

If P_{src} and P_{snk} intersect, let z denote their common vertex which lies "closest" to C, as shown on the right above. Notice that every vertex shared by P_{src} and P_{snk} lies outside C, since P_{src} and P_{snk} are shortest paths and we assumed $x \neq y$. In that case, we have found both a $\mathrm{D}\Psi$ and a $\mathrm{D}\Psi^{\mathrm{R}}$. If either of these subdivisions is bare in G, the claim follows, so assume that G contains bypasses for either subdivision. In particular, G contains a bypass of the $\mathrm{D}\Psi$. The pegs of Ψ in G are src, z, x, and y. First, we observe that G contains no bypass from any peg of Ψ to src, since $\mathrm{d}^-(src) = 0$. If the bypass we assume to exist is a (src, x)-bypass, we find $\mathsf{Q} \preccurlyeq G$, and if it is a (src, y)-bypass, we find $\mathsf{C} \preccurlyeq G$. If the bypass is from x to z, we find $\mathsf{C}^{\mathrm{R}} \preccurlyeq G$, and if we assume a (z, y)-bypass, $\mathsf{C}^{\mathrm{R}} \preccurlyeq G$ follows, too. This exhaust all cases for Ψ. The argument is symmetric for Ψ^{R}, so the claim follows for these cases.

2. If $x = y$, we denote this vertex simply x. Recall that C is a proper cycle, hence it contains further vertices. Let $z \in V_C \setminus x$ be incident to an out-arc $a = zk \notin A_C$; the argument is symmetric for an in-arc. Since G is p-normal, a is not parallel to the out-arc of z that belongs to C. Moreover, since we chose C as a minimal proper cycle, a is no chord of C, hence k lies outside C. Certainly, $k \neq src$ holds, since $\mathrm{d}^-(src) = 0$. If k lies on P_{src}, G contains a $\mathrm{D}\Psi$. The claim then follows just as in the previous case. If k lies on P_{snk}, then $\mathsf{C}^{\mathrm{R}} \preccurlyeq G$ follows. However, if k lies anywhere else in G, G contains an out-kebab, as well as an inout-kebab. Then the claim follows again from Lem. 7.2.9.

   Finally assume that every $z_i \in V_C \setminus x$ is incident to only two proper arcs. These must be its in-arc and out-arc in C. Now Prop. 7.2.7 implies that each z_i carries a loop. Since G is ℓ-normal, this also means that every z_i is a guard. However, the only vertices z_i might possibly guard, are the $z_j \in V_C \setminus \{x, z_i\}$. But since x already guards C by construction, and no z_i has a predecessor or successor outside C, this contradicts Prop. 6.1.2.

Each case implies a (bare) minor of G as claimed, or leads to a contradiction. Since the cases are exhaustive, the claim follows. □



(a) G_{C} (b) G_{Q} (c) G_{Ψ}

Figure 7.11: Hammocks witnessing the indispensability of C, Q, and Ψ in the minor-characterization of **SPL**.

We combine the results found in this section to finally get an alternative characterization of **SPL** by means of forbidden subgraphs.

Theorem 7.2.11. *Let G be a hammock. Then*

$$G \in \mathbf{SPL} \quad \textit{iff} \quad F \not\preccurlyeq G \textit{ for } F \in \mathbf{F} \textit{ and } K \not\preccurlyeq_{\mathrm{b}} G \textit{ for } K \in \mathbf{K}.$$

Proof. Let $G \in \mathbf{SPL}$, then G is a hammock. According to Lem. 7.1.1 G is **F**-free, while Lem. 7.1.5 states that G is free of bare **K**-minors.

Conversely, let $G \notin \mathbf{SPL}$ for a hammock G. Then Thm. 6.1.8 yields $\mathrm{R}(G) \neq \mathsf{P}_1$. If $\mathrm{R}(G)$ is acyclic, Valdes' theorem (Thm. 7.1.4) states $\mathsf{N} \preccurlyeq \mathrm{R}(G)$. If $\mathrm{R}(G)$ contains cycles, then Lem. 7.2.10 states $F \preccurlyeq \mathrm{R}(G)$ for some $F \in \mathbf{F}$ or $F \preccurlyeq_{\mathrm{b}} \mathrm{R}(G)$ for some $F \in \mathbf{K}$. If G is in normal form already, $G = \mathrm{R}(G)$, the claim follows immediately. Otherwise, we use that the existence of a (bare) DF is invariant under reduction, as stated in Lem. 7.2.2 for $F \in \mathbf{F}$ and in Lem. 7.2.4 for $F \in \mathbf{K}$. With these properties, induction on the length of the reduction from G to $\mathrm{R}(G)$ yields the claim. □

To conclude this chapter, we observe that the characterization of **SPL** by forbidden minors is not redundant: For each $F \in \mathbf{F} \cup \mathbf{K}$ we find a hammock $G_F \notin \mathbf{SPL}$ that witnesses F's indispensability from $\mathbf{F} \cup \mathbf{K}$. Each G_F satisfies $F \preccurlyeq_{\mathrm{b}} G_F$ and $F' \not\preccurlyeq G_F$ for $F' \in (\mathbf{F} \cup \mathbf{K}) \setminus F$. Since N is a hammock by itself and $F \not\preccurlyeq \mathsf{N}$ for $F \in \mathbf{F} \cup \mathbf{K}$, we find $G_{\mathsf{N}} = \mathsf{N}$. The analogous argument shows that $G_{\Phi} = \Phi$ holds. For the remaining graphs the witnessing graphs are shown in Fig. 7.11; notice that the witnesses for C^{R} and Ψ^{R} are the duals of the witnesses for C resp. Ψ.

This concludes our treatment of the obstruction set characterization of **SPL**. Let us state once more that the characterization we found is consistent with

the respective one for **SP**. As a matter of fact, a result of the same type is known for a nontrivial superset of **SPL**. The class in question is investigated in the context of structured programming, otherwise known as GOTO-free programming. On a graph theoretic level, it can be defined (in our terms) as the class generated from P_1 by the spl-expansion, together with an operation which replaces a constriction with the graph Φ.

The resulting class obviously allows Φ as a bare minor. Moreover, within the class of hammocks this larger class admits a forbidden minor characterization, too, given by the set **F** alone [23, 42]. Intuitively, this implies that the "part" of our characterization which consists of **K** necessitates from the absence of Φ as a bare minor. Nevertheless, as we just discussed, the characterization is minimal, in the sense that no member of **K** can be left out. However, this fact follows relative to the notion of a "bare" minor; in the light of the superclass of **SPL** and its characterization, it might be well possible to obtain an obstruction set characterization of **SPL** that consists of fewer graphs wrt. a different minor notion on directed graphs.

8 SPL-Graphs and Regular Expressions

In this chapter we investigate how spl-graphs can be reversibly encoded. An "encoding" of an spl-graph is meant to be a term, i.e., a linear entity, that can be efficiently translated to a unique member of **SPL**. It is not particularly surprising that this is possible in the first place, since **SPL** is a recursively constructed class of graphs. Any member of such a class is implicitly characterized by a decomposition tree, which carries the information on how to construct the considered member by using the recursive rules of the definition. A decomposition tree, in turn, can be written in term form quite naturally. The characterization by decomposition trees often allows efficient solutions of problems that are hard on arbitrary graphs [5, 47, 6].

We choose regular expressions to encode spl-graphs. More precisely, we interpret the decomposition tree of an spl-graph as the parse tree of an expression, which yields the encoding immediately. This choice provides us with several results that go beyond the mere property of conveying the graph structure. In particular, we will find an efficient conversion of EFAs with spl-structure to expressions that are linear in EFA size. Notice, however, that decomposition trees are generally not unique. This is also the case for spl-graphs, resp. for the expressions encoding these graph. However, the ambiguity is restricted to associativity and commutativity of regular operators, which is not critical.

Considering the dual operation of encoding an spl-graph, we further show that any reduced expression encodes a unique spl-graph. This graph can be reconstructed from the expression by an adapted fragment of the ARS we gave in Ch. 4 for the construction of FAs from expression. In fact, we treat EFAs and spl-graphs in a unified framework; in particular, we will identify hammocks with trim normalized automata.

Definition 16. A *labeled graph* is a triple $(G, \mathcal{A}, l)$, where G is a graph, $\mathcal{A}$ is an alphabet and l is a map $l : A_G \to \mathrm{RE}_{\mathcal{A}}$, called the *labeling*. The image of an arc under l is the *label* of the arc. A labeling is *p-injective*, if distinct parallel arcs of G are mapped to distinct labels.

When referring to the structural properties of a labeled graph $G = (G', \mathcal{A}, l)$, we are actually considering G'. A graph as it was considered in the previous chapters is also called an *unlabeled* graph. For the unified treatment of unlabeled and labeled graphs, the unlabeled graph G is associated with the labeled graph (G, A_G, Δ_{A_G}). The labeling Δ_{A_G} is the identity on A_G, i.e., each arc is considered as a letter, resp. a trivial expression. In other words, each arc in this graph is labeled with "itself", which is clearly a p-injective labeling.

Recall that the graph underlying an EFA $E = (Q, \mathcal{A}, \delta, I, F)$, is defined as $\mathrm{G}(E) = (Q, \delta, \pi_1^3, \pi_3^3)$ with tail map $\pi_1^3 : (p, r, q) \mapsto p$ and head map $\pi_3^3 : (p, r, q) \mapsto q$. We extend this notion to labeled graphs by also taking the label of each transition into account.

Definition 17. Let E be an EFA over $\mathcal{A}$, then the *graph interpretation* of E is the labeled graph $\mathrm{G}_l(E)$, defined as

$$\mathrm{G}_l(E) := (\mathrm{G}(E), \mathcal{A}, \pi_2^3).$$

Observe that by switching to the graph interpretation of an EFA, we discard information about the initial and final states of the automaton. Any other feature, however, is preserved and can be restored from the graph interpretation. Still, this implies that we generally cannot reconstruct the language accepted by an EFA from its graph interpretation. We will see that this does not pose a problem if trim normalized EFAs are considered. To this end, we define a counterpart notion of graph interpretation.

Definition 18. Let $G = (G', \mathcal{A}, l)$ be a labeled graph and let $I = \{x \in V_{G'} \mid \mathrm{d}^-_{G'}(x) = 0\}$ and $F = \{x \in V_{G'} \mid \mathrm{d}^+_{G'}(x) = 0\}$. The *automaton interpretation* of G is the EFA $\mathrm{FA}(G)$, defined as

$$\begin{aligned} \mathrm{FA}(G) &:= (V_{G'}, \mathcal{A}, \delta, I, F), \text{ where} \\ \delta &:= \{(t_{G'}(a), l(a), h_{G'}(a)) \mid a \in A_{G'}\}. \end{aligned}$$

The language accepted by the graph G is now defined as $\mathrm{L}(G) := \mathrm{L}(\mathrm{FA}(G))$.

Observe the sets of initial and final states of an EFA are allowed to be empty, so the automaton interpretation of a labeled graph is well defined. This also holds for the language accepted by a labeled graph.

In the following, we consider the homomorphic extension of a labeling to sequences of arcs. For $w = a_1 \dots a_n$, $a_i \in A$, we set

$$l(w) = l(a_1 \dots a_n) := l(a_1) \dots l(a_n),$$

wherein each label becomes a factor in a product.

Proposition 8.0.12. *Let $G = (G', \mathcal{A}, l)$ be a labeled graph. If w is an (x, y)-walk in G, then each word in $\mathrm{L}(l(w))$ carries x to y in $\mathrm{FA}(G)$.*

Proof. Straightforward induction over the length of the walk, using the homomorphic extension of the labeling. □

If we consider an unlabeled graph, i.e., if we label each arc with itself, a stronger — although unsurprising — result holds.

Proposition 8.0.13. *Let G be an unlabeled graph, then w is an (x, y)-walk in G iff w carries x to y in $\mathrm{FA}(G)$.*

Proof. The homomorphic extension of the identity on arcs equals the identity on walks. □

The motivation behind choosing the vertices with in-degree (out-degree) as initial (final) states in the automaton interpretation becomes clear from the following properties.

Proposition 8.0.14.

1. *If H is a labeled hammock, then $\mathrm{FA}(H)$ is a trim normalized EFA.*
2. *If E is a trim and normalized EFA, then $\mathrm{G}(E)$ is a labeled hammock.*
3. *If G is a labeled graph with a p-injective labeling, then $\mathrm{G}(\mathrm{FA}(G)) = G$.*
4. *If E is a trim and normalized EFA, then $\mathrm{FA}(\mathrm{G}(E)) = E$.*

Proof.

1. Let $H = (H', \mathcal{A}, l)$ with $(H', src, snk) \in \mathbf{H}$, then the initial and final states of $\mathrm{FA}(H)$ are the singleton sets $\{src\}$ and $\{snk\}$, respectively. Moreover, for $x \in V_H$ there is a (src, snk)-path P and an (x, snk)-path P' in H. As each path is a special kind of walk, Prop. 8.0.12 applies, yielding that x is accessible and co-accessible in $\mathrm{FA}(H)$. Therefore $\mathrm{FA}(H)$ is trim and normalized.
2. Let E be trim and normalized, with initial state q_i and final state q_f. Since every state is accessible and co-accessible in E, it lies on a (q_i, q_f)-path in $\mathrm{G}(E)$. With $\mathrm{d}^-_{\mathrm{G}(E)}(q_i) = 0$ and $\mathrm{d}^+_{\mathrm{G}(E)}(q_f) = 0$ we find that $\mathrm{G}(E)$ is a hammock with source q_i and sink q_f.

3. The claim follows immediately from the definitions of a graph interpretation and an automaton interpretation. The requirement that the labeling is p-injective is necessary, since otherwise, parallel arcs with coinciding labels are mapped to the same transition in FA(G).

4. Let E be trim and normalized with initial state q_i and final state q_f. Then G(E) is a hammock with source q_0 and sink q_f. According to its definition, the automaton interpretation of G(E) again has q_0 as an initial and q_f as a final state.

□

Supported by Prop. 8.0.14, we do sometimes not clearly distinguish trim normalized EFAs and labeled hammocks with p-injective labelings. In particular, we formulate the results in this chapter in the framework of labeled hammocks, yet apply them to trim normalized FAs.

8.1 Encoding Graphs by Expressions

In order to encode members of **SPL**, we augment the spl-reductions to labeled hammocks. Recall from Sec. 6.1 that reducing an spl-graph G to P_1 is equivalent to finding a sequence of expansions from P_1 to G backwards. By using labels in the reduction process, information about this sequence is stored in the labels. More specifically, a label stores the recursive structure of the subgraph that emerges from the arc carrying the label. We associate series reductions with products, parallel reductions with sums and loop reductions with iterations. The universe of the resulting ARS is the class of labeled hammocks over $\mathcal{A}$, defined as

$$\mathbf{H}_{\mathcal{A}} := \{(H, \mathcal{A}, l) \mid H \in \mathbf{H}\}.$$

The path of length one, P_1, can hold only a single label, which we convey in formal notation. Recall that P_1 is the axiom of **SPL**, so we define a *labeled axiom* to be any labeled graph

$$\mathsf{P}_1[r] := (\mathsf{P}_1, \mathcal{A}_r, l),$$

where $\mathcal{A}_r$ is the least alphabet for which r is defined, and $l(a) = r$ for the sole arc a of P_1.

Definition 19. The relations $\blacktriangleleft.$, $\blacktriangleleft_+$, and $\blacktriangleleft_*$, called *product encoding*, *sum encoding*, and *star encoding*, are defined on $\mathbf{H}_\mathcal{A}$ as described below. In the following let $G_1 = (G_1', \mathcal{A}, l_1)$ and $G_2 = (G_2', \mathcal{A}, l_2)$ be members of $\mathbf{H}_\mathcal{A}$.

- The relation $G_1 \blacktriangleleft. G_2$ holds if the following is satisfied:
  1. The vertex y is simple in G_1', and $G_1' \overset{s}{\Leftarrow} G_2$ holds due to $G_2 = (G_1' - y) \cup xz$.
  2. The arcs incident to y in G_1' are $a_1 = xy$ and $a_2 = yz$, with labels $l_1(a_1) = s$ and $l_1(a_2) = t$.
  3. The labeling in G_2 is $l_2(a) = st$ and $l_2(a') = l_1(a')$ for $a \neq a'$.
- The relation $G_1 \blacktriangleleft_+ G_2$ holds if the following is satisfied:
  1. The arcs a, a' are parallel in G_1', and $G_1' \overset{p}{\Leftarrow} G_2'$ holds due to $G_2' = G_1' \setminus a$.
  2. The labels of a and a' in G_1 are $l_1(a) = s$ and $l_1(a') = t$.
  3. The labeling in G_2 is $l_2(a') = s + t$ and $l_2(a'') = l_1(a'')$ for $a'' \notin \{a, a'\}$.
- The relation $G_1 \blacktriangleleft_* G_2$ holds if the following is satisfied:
  1. G_1' contains a loop $a = xx$, and $G_1' \overset{\ell}{\Leftarrow} G_2'$ holds due to $G_2' = G_1' \setminus a \ll x \gg$.
  2. The label of a in G_1 is $l_1(a) = s$.
  3. The labeling in G_2 is $l_2(x_1x_2) = s^*$ and $l_2(a') = l_1(a')$ for $a \neq x_1x_2$.

More generally, the *encoding* relation is defined as $\blacktriangleleft := \blacktriangleleft. \cup \blacktriangleleft_+ \cup \blacktriangleleft_*$.

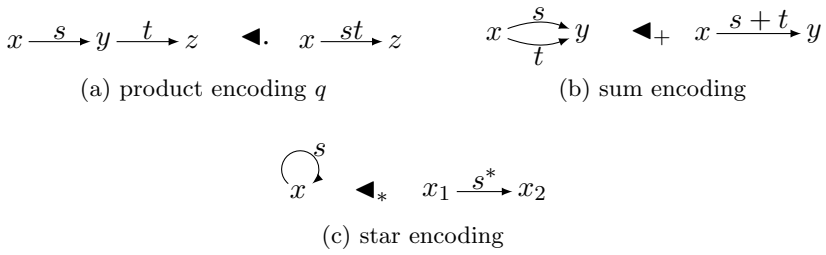
The local differences for labeled hammocks are shown in Fig. 8.1 for each encoding. Encodings are the rules of the ARS $\mathfrak{E}$, which is defined as

$$\mathfrak{E} := \langle \mathbf{H}_\mathcal{A}, \blacktriangleleft., \blacktriangleleft_+, \blacktriangleleft_* \rangle.$$

It becomes clear from their definition that the encoding relations merely augment the spl-reductions by handling arc labels. Consequently, for labeled graphs $G_i = (G_i', \mathcal{A}, l_i)$, we find

$$G_1 \blacktriangleleft G_2 \blacktriangleleft \cdots G_n \text{ iff } G_1' \Leftarrow G_2' \Leftarrow \cdots G_n'.$$

Recall from Sec. 6.1 that the ARS $\mathfrak{R}$ of spl-reductions is terminating and locally confluent. Therefore, $\mathfrak{E}$ is certainly terminating, too, and admits normal forms that are at least structurally unique. However, the labeling of a normal form



(a) product encoding q (b) sum encoding

(c) star encoding

Figure 8.1: Labeled reductions, or encodings, on $\mathbf{H}_{\mathcal{A}}$.

in $\mathfrak{E}$ is not necessarily unique. More precisely, it is unique up to algebraic identities on regular expressions. These are associativity of products and sums, i.e., $(rs)t \equiv r(st)$, resp. $(r + s) + t \equiv r + (s + t)$, and commutativity of sums, $r+s \equiv s+r$. In the following, we refer to these properties loosely as associativity and commutativity of (operators in) expressions, while keeping in mind that commutativity refers to sums only. We write

$$r_1 =_{\mathrm{AC}} r_2$$

if r_1 and r_2 are identical modulo associativity and commutativity[1]. To prove the claim above, we take a step back and establish local confluence for encodings modulo $=_{\mathrm{AC}}$ separately.

Lemma 8.1.1. *The ARS $\mathfrak{E}$ is locally confluent modulo $=_{\mathrm{AC}}$ in labels.*

Proof. Let $G = (G', \mathcal{A}, l)$ and $G_i = (G'_i, \mathcal{A}, l_i)$, for $i \in \mathbb{N}$, be labeled hammocks. We need to show the following: given a pair of encodings $G \blacktriangleleft_{\mathrm{i}} G_1$ and $G \blacktriangleleft_{\mathrm{j}} G_2$, for $\mathrm{i}, \mathrm{j} \in \{\cdot, +, *\}$, there are G_3, G_4 s.t. encodings $G_1 \blacktriangleleft^{\star} G_3$ and $G_2 \blacktriangleleft^{\star} G_4$ exist, with $G'_3 = G'_4$ and $l_3(a) =_{\mathrm{AC}} l_4(a)$ for each arc a.

A thorough proof of this requires to consider every combination $\mathrm{i}, \mathrm{j} \in \{\cdot, +, *\}$. All cases are straightforward: most of the work is identical to that in the proof of Lem. 6.1.6, although plenty of cases arise when considering commutative and associative combinations. Therefore, we only prove a first case in full rigor and argue rather superficially in the remaining ones.

Assume $G \blacktriangleleft. G_1$ and $G \blacktriangleleft. G_2$, meaning that for $i \in \{1, 2\}$ the following holds:

1. y_i is simple in G', for $i \in \{1, 2\}$ with predecessor x_i and successor z_i,

[1] We omit a formal definition of this relation, which is easily given inductively.

2. $G_i = (G - y_i) \cup a_i$ for a new $x_i y_i$-arc a_i, and
3. $l_i(a_i) = l(x_i y_i) l(y_i z_i)$.

If $y_1 \notin \{x_2, z_2\}$, then either $y_1 = y_2$, which yields the claim trivially, or the arcs incident to y_1 and y_2 are disjoint. In the latter case, either G_i is further encoded to $H = (H', \mathcal{A}, l_H)$, where

$$H' = (G' - \{y_1, y_2\}) \cup \{a_1, a_2\},$$

and $l_H(a_i) = l(x_i y_i) l(y_i z_i)$. In that case, setting $G_3 = G_4 := H$ satisfies the claim.

If we assume $y_1 = x_2$ (the case $y_1 = z_2$ is symmetric), then $z_1 = y_2$ follows, i.e., the $y_1 z_1$-arc and the $x_2 y_2$-arc of G are identical. Let $l(x_1 y_1) = r$, $l(y_1 z_1) = l(x_2 z_2) = s$ and $l(y_2 z_2) = t$, as shown below:

$$x_1 \xrightarrow{\ r\ } y_1/x_2 \xrightarrow{\ s\ } z_1/y_2 \xrightarrow{\ t\ } z_2$$

In the following, we drop the vertex "names" x_2 and z_1. If expressions are written according to their strict definition, i.e., fully parenthesized, the label of a_1 in G_1 is $l_1(a_1) = l(x_1 y_2) = (rs)$, while that of a_2 in G_2 is $L_2(a_2) = l_2(y_1 z_2) = (rs)$. The following sketch shows this part in G_1 on the left and that in G_2 on the right.

$$x_1 \xrightarrow{\ (rs)\ } y_2 \xrightarrow{\ t\ } z_2 \qquad\qquad x_1 \xrightarrow{\ r\ } y_1 \xrightarrow{\ (st)\ } z_2$$

Since y_2 is simple in G_1 and y_1 is simple in G_2, so either G_i admits for a further product encoding: $G_1 \blacktriangleleft. G_3$ due to $G_3' = (G_1' - y_2) \cup x_1 z_2$ with $l_3(x_1 z_2) = ((rs)t)$, and $G_2 \blacktriangleleft. G_4$ due to $G_4' = (G_2' - y_1) \cup x_1 z_2$ with $l_4(x_1 z_2) = (r(st))$. These graphs satisfy $G_3' = G_4'$, while for the labels, we find $l_3(x_1 z_2) =_{\mathrm{AC}} l_4(x_1 z_2)$ and $l_3(a) = l_4(a)$ for $a \neq x_1 z_2$, as claimed. This completes the first case.

Now let $G \blacktriangleleft. G_1$ and $G \blacktriangleleft_+ G_2$. Let $G_1' = (G' - y) \cup xz$ and $G_2' = G' \setminus a$. Assume that a is parallel to a' in G and that $l(a) = s$ and $l(a') = s'$. As x is certainly not incident to a, the two encodings do not interfere with each other, i.e., x is simple in G_2, and a is parallel to some further arc in G_1. This allows for encodings $G_1 \blacktriangleleft_+ G_3$ and $G_2 \blacktriangleleft. G_4$, due to $G_3' = G_1' \setminus a$ and $G_4' = (G_1' - y) \cup xz$. We know from Lem. 6.1.6 that $G_3' = G_4'$. The label of a coincides in G_3 and G_4, or it is $s + s'$ in G_3 and $s' + s$ in G_4, or vice versa. In each case, $l_3(a) =_{\mathrm{AC}} l_4(a)$ is satisfied.

Assume that the encodings are $G \blacktriangleleft. G_1$ and $G \blacktriangleleft_* G_2$. There is no way to introduce distinct labels if the two encodings are successively applied in either order. In this case, the claim follows immediately from Lem. 6.1.6.

For two sum encodings, $G \blacktriangleleft_+ G_1$ and $G \blacktriangleleft_+ G_2$, let $G'_i = G \setminus a_i$ for $i \in 1, 2$, and let a_i be parallel to a'_i in G. If a_1 and a_2 are not parallel to another, then a_1 is (still) parallel to a'_1 in G_2 and a_2 is parallel to a'_2 in G_1. The sum obvious sum encodings $G_1 \blacktriangleleft_+ G_3$ and $G_2 \blacktriangleleft_+ G_4$ yield labeled graph with $G'_3 = G'_4$ and $l_3(a'_1) =_{\mathrm{AC}} l_4(a'_2)$.

If the a_i are parallel to another for a pair of sum encodings, assume that the a_i are xy-arcs and let H denote the labeled graph derived from G by replacing all xy-arcs with a single arc a, labeled with the sum of all labels of xy-arcs in G. Let G_3 and G_4 denote the labeled graphs derived from G_1 and G_2 by sum encoding all the remaining xy-arcs, so $G_1 \blacktriangleleft^\star_+ G_3$ and $G_2 \blacktriangleleft^\star_+ G_4$. $G'_3 = G'_4$ is clear in this case. Moreover, we find $l_3(xy) =_{\mathrm{AC}} l_H(xy) =_{\mathrm{AC}} l_4(xy)$, thus also $l_3(xy) =_{\mathrm{AC}} l_4(xy)$.

The encodings $G \blacktriangleleft_+ G_1$ and $G \blacktriangleleft_* G_2$ again do not interfere with another, and their order can be swapped. In other words, there are always follow-up encodings $G_1 \blacktriangleleft_* G_3$ and $G_2 \blacktriangleleft_+ G_4$ s.t. $G_3 = G_4$.

Let finally $G \blacktriangleleft_* G_i$ due to $G'_i = (G' \setminus a_i) \ll x_i \gg$. If $x_1 = x_2$, then $a_1 = a_2$ follows, since loop reduction is not allowed for parallel loops. In that case, the statement follows trivially from $G_1 = G_2$. Otherwise, the order of the encodings is irrelevant, since either G_i allows for the "other" encoding, and $G_3 = G_4$ follows.

This covers all cases and the discussion is complete. □

Corollary 8.1.2. *Let $G = (G', \mathcal{A}, l)$ with $G' \in$ **SPL** and l p-injective. Then $G \blacktriangleleft^\star \mathsf{P}_1[r]$ follows, where r is unique modulo $=_{\mathrm{AC}}$.*

Proof. For $G \in$ **SPL** we have established $G \Leftarrow^\star \mathsf{P}_1$ in Ch. 6. If we consider the unlabeled graph G as a labeled graph where each arc is labeled with itself, we get $G \blacktriangleleft^\star \mathsf{P}_1[r]$ for some expression r. Following Lem. 8.1.1, $\mathsf{P}_1[r]$ is unique modulo $=_{\mathrm{AC}}$ in labels. Since r is such a label, the claim follows. □

This brings us to the actual description of graphs with spl-structure by means of regular expressions. Recall that $\mathrm{red}(r)$ denotes the reduced equivalent of r, and that $\Leftarrow r$ either equals one of $\varnothing$ and ε, or is free of both.

Definition 20. Let $G = (G', \mathcal{A}, l)$ with $G' \in \mathbf{SPL}$ and l p-injective. An *encoding expression* of G, denoted $\mathrm{RE}(G)$, is any expression $\mathrm{red}(r)$ s.t. $G \blacktriangleleft^\star \mathsf{P}_1[r]$. Likewise, an encoding expression of an spl-EFA E, denoted $\mathrm{RE}(E)$, is any expression r s.t. $r = \mathrm{RE}(\mathrm{G}(E))$.

We usually call an encoding expression just an "encoding". It will always be clear from the context whether we refer to a rule of $\mathfrak{E}$ or an expression derived through rewritings in $\mathfrak{E}$.

Notice that for fixed r, the expression $\mathrm{red}(r)$ is unique. Therefore, the encoding of an spl-graph is unique modulo $=_{\mathrm{AC}}$. This is no severe drawback. It is easy to define a normal form on expressions where all products and sums are written in left associative form and the operands of sums are recursively ordered in a lexicographic fashion. Any expression can be put into this normal form in time $\mathcal{O}(n \log n)$ where n is the alphabetic width of the expression.

An algorithm for computing an encoding is derived by generalizing Alg. 4 to graphs with labeled arcs. Notice that common implementations of dynamic graphs support labels already [48, 34]. Moreover, the encoding rules can be implemented without a significant overhead. This required that labels, expressions, are not treated as strings but — by their recursive nature — as trees. This avoids costly operations on strings, which would incur at least a logarithmic overhead.

We briefly outline one way to deal with expressions as labels in an implementation. Therein an arc is associated with a pointer to the parse of an expression, which is supposed to be the label of this arc. This tree is stored separately. With each encoding step either two trees are merged, in the case of product and sum encoding, or a new root with label $*$ is added, for star encoding. Generally, the label of an arc that is manipulated becomes a subtree in a bigger tree. If these trees are implemented adequately, these operations take constant time each. Reducing an expression is easily implemented on the parse of the expression in linear time.

For all practical purposes, $\mathfrak{E}$ can also be considered as an ARS on trim normalized EFAs. The only difference between a graph with p-injective labeling and an EFA is the way how initial and final states are determined. The equivalence of hammocks and trim normalized EFAs was shown in Prop. 8.0.14. We continue by treating labeled hammocks as language acceptors in the following

Proposition 8.1.3. *If* $G \blacktriangleleft H$, *then* $\mathrm{L}(G) = \mathrm{L}(H)$.

Proof. The source or sink of G is not removed upon encoding, thus the sources of G and H coincide, as do the sinks. Hence, the trim normalized EFAs $\mathrm{FA}(G)$ and $\mathrm{FA}(H)$ have the same initial and final states. Showing that w carries the initial to the final state in $\mathrm{FA}(G)$ iff w carries the initial to the final state in $\mathrm{FA}(H)$ is straightforward. □

For unlabeled spl-graphs, we find that the set of walks from the source to the sink is denoted by a regular expression, namely, any encoding of the graph.

Corollary 8.1.4. *Let $(G, src, snk) \in \mathbf{SPL}$. Then w is a (src, snk)-walk in G iff $w \in \mathrm{L}(\mathrm{RE}(G))$.*

Proof. According to Prop. 8.0.13, w is a (src, snk)-walk in G iff $w \in \mathrm{L}(G)$. Since we assume $G \in \mathbf{SPL}$, G can be reduced to P_1. This implies $G \blacktriangleleft^\star \mathsf{P}_1[r]$, for some expression r. From Prop. 8.1.3 follows $\mathrm{L}(G) = \mathrm{L}(\mathsf{P}_1[r]) = \mathrm{L}(r)$, which yields the claim. □

The Cor. 8.1.4 is a first justification to interpret the decomposition tree of an spl-graph as an expressions, and to associate series, parallel, and loop properties with the regular operators as we did. As an important "byproduct", we obtain an efficient method to convert EFAs with spl-structure to regular expression.

Theorem 8.1.5. *Let E be an spl-EFA and let $r = \mathrm{RE}(E)$. Then $\mathrm{L}(r) = \mathrm{L}(E)$, and r can be constructed in time $\mathcal{O}(|Q_E|^3)$.*

Proof. For an spl-EFA E we find $\mathrm{G}(E) \blacktriangleleft^\star \mathsf{P}_1[r']$ with $\mathrm{L}(r') = \mathrm{L}(\mathrm{G}(E)) = \mathrm{G}(E)$. Managing labels does not cause an overhead of encodings over reductions, so the running time of this computation equals that of constructing P_1 from $\mathrm{G}(E)$. Following Prop. 6.2.7, this can be done in time $\mathcal{O}(|V_{\mathrm{G}(E)}|^3)$, or equivalently $\mathcal{O}(|Q_E|^3)$.

For the reduction of r' to r we only need to consider ε-reduction, since r' is naturally $\varnothing$-free. The number of subsequent rewritings of r' equal the number of ε-positions in r'. Since every ε-position stems from an ε-transition in E, the number of operations necessary for reduction is in $\mathcal{O}(|Q_E|)$. This second step does not change the asymptotic running time, so the claim follows. □

A case of particular importance for Thm. 8.1.5 is that of spl-FAs. The theorem implies the efficient conversion of such FAs into expressions. This generalizes a recent result by Moreira & Reis [38], who gave an efficient conversion of FAs

with series parallel structure to expressions that are linear in the size of the FA. In the following, we show that the same is true for FAs with series parallel loop structure; our findings are brought together in Thm. 8.1.8.

Proposition 8.1.6. *Let A be an spl-FA. Then $|\operatorname{RE}(A)|_{\mathcal{A}} \leq |\delta_A|$.*

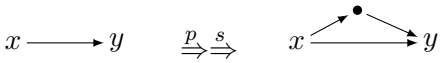
Proof. It is evident from Fig. 8.1 that the sum of alphabetic widths of all transitions in an EFA / labeled hammock remains constant upon encoding. In an FA, this number equals the number of transitions, which implies the claim. □

According to our definition of the size of an automaton, we also need to take the number of states into account. To this end we bound the alphabetic width of an encoding once more.

Lemma 8.1.7. *Let A be an spl-FA over $\mathcal{A}$. Then $|\operatorname{RE}(A)|_{\mathcal{A}} < 2|Q_A|(|\mathcal{A}_A|+1)$.*

Proof. First, we bound the number of transition in an spl-FA relative to the number of states. We first assume that the FA is free of parallel transitions, i.e., normal wrt. sum-encoding, and treat parallel transitions later.

On the graph level, we examine how a p-normal spl-graph is derived from P_1. Obviously, P_1 is p-normal. Assume that G is p-normal and let $G \Rightarrow H$. For $G \overset{s}{\Rightarrow} H$ and $G \overset{\ell}{\Rightarrow} H$, we find that H is p-normal, too. To enclose all p-normal spl-graphs, we also need to consider intermediate p-expansions in the construction. So for p-normal G let $G \overset{p}{\Rightarrow} H$, which is clearly not p-normal. Let a and a' denote the unique pair of parallel arcs in H. To get a p-normal graph from H, further p-expansion is obviously useless. Also, ℓ-expansion can not be applied to a, a', neither arc is a constriction. However, applying s-expansion to either arc yields a p-normal spl-graph again. So we assume a combination of p- and s-expansion, i.e., the replacement of an arc by a so-called "transitive triple", as sketched below:



It is easy to see that this actually yields all p-normal spl-graphs, basically by reverting the constructive process described above. We only sketch this direction. Let $G \in \mathbf{SPL}$ be p-normal, then either $G = \mathsf{P}_1$ holds, or s- or ℓ-reduction applies to G. In the first case, at most one pair of parallel arcs

may emerge; a follow-up p-reduction makes the graph p-normal again. This is exactly the reversal of the combined expansion shown above. In the second case, parallel arcs do not emerge from reduction.

For $c \in \{s, p, \ell\}$, let $\#_c$ denote the number of c-expansions in the construction of a p-normal spl-graph G from $\mathsf{P_1}$. The preceding discussion implies that every p-expansion is followed by s-expansion, so we have $\#_p \leq \#_s$. If we count the number of arcs and vertices introduced with each decoding/expansion (cf. Fig. 8.2), starting with $|A_{\mathsf{P_1}}| = 1$ and $|V_{\mathsf{P_1}}| = 2$, we find

$$\begin{aligned} |A_G| &= 1 + \#_s + \#_p \leq 1 + 2\#_s, \text{ and} \\ |V_G| &= 2 + \#_s - \#_\ell. \end{aligned}$$

We solve the lower equation for $\#_s$ and substitute the result for $\#_s$ in upper inequation to find that the number of arcs in a p-normal spl-graph is bounded as

$$|A_G| \leq 2|V_G| - 2\#_\ell - 3 < 2|V_G|.$$

Considering arbitrary $G \in \mathbf{SPL}$, observe that is irrelevant at which point in the rewriting $\mathsf{P_1} \Rightarrow^\star G$ the parallel arcs of G are introduced. That is, we may assume that G is derived by

$$\mathsf{P_1} \Rightarrow^\star G' \overset{p}{\Rightarrow}^\star G,$$

where G' is p-normal.

While the number of parallel arcs in spl-graphs is unbounded, this is different for spl-FAs. For states p and q, there may be at most $|\mathcal{A}| + 1$ transitions from p to q, since we allow for ε-transitions. Let A be such an automaton, then the bound derived for p-normal spl-graphs, is reinterpreted for spl-FA and corrected by parallel transitions, to yield

$$|\delta_A| \leq (|\mathcal{A}_A| + 1)(2|Q_A| - 5) < 2|Q_A|(|\mathcal{A}_A| + 1).$$

The claim now follows with Prop. 8.1.6. □

The bound in Lem. 8.1.7 is interesting in itself, since the number of states and the alphabetic width are among the more common measures of automata and expression complexity/size (cf. introductory section of Ch. 3). Nevertheless, we further bound the size of an expression relative to the size of an spl-FA wrt. the notions that were used in Chs. 3 and 5 to refer to the size of an FA or expression.

Theorem 8.1.8. *For an spl-FA A an equivalent expression r can be computed in $\mathcal{O}(\max\{|Q_A|^3, |\delta_A|^2\})$. The size of r is bounded in the size of A as*

$$|r| < 6|Q_A|(|\mathcal{A}_A| + 1).$$

Proof. We set

$$r := \mathrm{RE}(A)^\bullet,$$

then $|r|_{\mathcal{A}} = |\mathrm{RE}(A)|_{\mathcal{A}}$ holds trivially. As shown in Lem. 8.1.7, the expression interpretation of A satisfies $|\mathrm{RE}(A)|_{\mathcal{A}} \leq 2|Q_A|(|\mathcal{A}_A|+1)$. Following Thm. 3.2.10, the overall size of r is bounded within three times its alphabetic width, so we get

$$|r| \leq 3|r|_{\mathcal{A}} = 3|\mathrm{RE}(A)|_{\mathcal{A}} < 6|Q_A|(|\mathcal{A}_A| + 1),$$

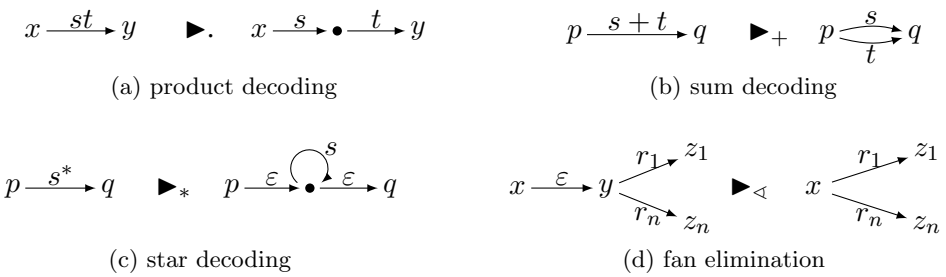
as claimed.

The time needed to compute r from A is the time for labeled reduction and computation of the SSNF. The latter computation is linear in $|\mathrm{RE}(A)|$, so the overall running time is dominated by computing the expression interpretation. The complexity of the latter equals that of reducing $\mathrm{G}(A)$ to P_1, which can be done in $\mathcal{O}(\max\{|V_{\mathrm{G}(A)}|^3, |A_{\mathrm{G}(A)}|^2\})$, as was demonstrated in Prop. 6.2.7 for Alg. 4. □

8.2 Decoding Expressions to Graphs

We have previously claimed that the encoding expression of an spl-graph contains the recursive structure of the graph. In this section we give the construction to recover this structure, i.e., we consider the construction of an spl-graph from its encoding. This construction will be defined for arbitrary reduced expressions; we will show that any reduced expression encodes an spl-graph. Recall from Sec. 4.4 that every expression can be converted to a smaller equivalent expression that is reduced; recall further that ε-freeness of reduced expression is partially due to the syntactic convention that $r + \varepsilon$ is written as $r^?$.

We consider this second construction in form of a further ARS on labeled hammocks. Unlike for the encoding relation, it does not suffice to augment the graph-theoretic relations — the spl-expansions, in this case — with labels. This would not guarantee that the system produces graphs for all inputs in a unified manner. While we do consider labeled spl-expansions, we need a further rule to remove ε-transitions that appear from labeled loop-expansions.



(a) product decoding (b) sum decoding

(c) star decoding (d) fan elimination

Figure 8.2: Decodings and fan elimination on $\mathbf{H}_{\mathcal{A}}$

As it turns out, we can use a fragment of the ARS which we considered in Ch. 4 for the construction of FAs from arbitrary expressions. More specifically, we restrict ourselves to product-, sum-, and star-expansion, as well as fan-elimination. However, these rules are defined on EFAs; still, as we have seen in Prop. 8.0.14, we can at least identify trim normalized EFAs with labeled hammocks.

For a sound formal treatment, we introduce new relations on labeled hammocks, which indicate that a certain relation holds for the automaton interpretations of these graphs. To this end, for labeled hammocks G and H, we write

- $G \blacktriangleright. H$ if $\mathrm{FA}(G) \Rightarrow. \mathrm{FA}(H)$
- $G \blacktriangleright_+ H$ if $\mathrm{FA}(G) \Rightarrow_+ \mathrm{FA}(H)$
- $G \blacktriangleright_* H$ if $\mathrm{FA}(G) \Rightarrow_* \mathrm{FA}(H)$
- $G \blacktriangleright_\lhd H$ if $\mathrm{FA}(G) \Rightarrow_\lhd \mathrm{FA}(H)$

This yields the decoding ARS

$$\mathfrak{D} := \langle \mathbf{H}_{\mathcal{A}}, \blacktriangleright., \blacktriangleright_+, \blacktriangleright_*, \blacktriangleright_\lhd \rangle.$$

The decoding rules are shown in Fig. 8.2. We set $\blacktriangleright := \blacktriangleright. \cup \blacktriangleright_+ \cup \blacktriangleright_* \cup \blacktriangleright_\lhd$.

Lemma 8.2.1. *The ARS $\mathfrak{D}$ is locally confluent.*

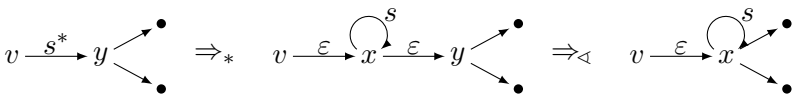
Proof. The proof is standard by showing that $F \blacktriangleright G_1$ and $F \blacktriangleright G_2$ implies $G_i \blacktriangleright^\star H$ for some labeled graph H and $i \in \{1, 2\}$. We have shown the corresponding property for $\Rightarrow.$, $\Rightarrow_+$, $\Rightarrow_*$, and $\Rightarrow_\lhd$ on EFAs in Prop. 4.2.3, Lems. 4.2.4 and 4.2.6. This result carries over to labeled graphs and decodings, since the decoding ruled are defined by the EFA relations. □

Proposition 8.2.2. *Every intermediate graph in the process of decoding a reduced expression is a labeled spl-graph.*

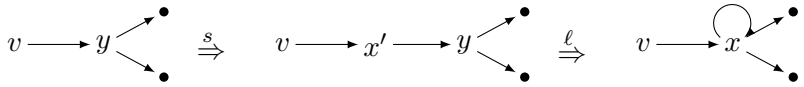
Proof. A decoding step inflicts product, sum, or star expansion, or fan elimination on the graph structure, so we prove that these rewritings yield spl-graphs from P_1. The labeled axiom, resp. the primal EFA of an expression, is a labeled spl-graph. Assume that $G = (G', \mathcal{A}, l_G)$ for $G \in \mathbf{SPL}$ and let $G \Rightarrow H$ for $H = (H', \mathcal{A}, l_H)$. It is obvious from Figs. 8.2 and 4.1 that $G \Rightarrow. H$ implies $G' \overset{s}{\Rightarrow} H'$ and that $G \Rightarrow_+ H$ implies $G' \overset{p}{\Rightarrow} H'$. In either case $H' \in \mathbf{SPL}$ follows.

The effects of star decoding on the structure of an EFA can be brought forward by spl-expansions, too. If $G \Rightarrow_* H$ holds, then H' is derived from G' by replacing an arc $a = xy$ with a new vertex z, and arcs $a_1 = xz$, $a_2 = zz$, and $a_3 = zy$. This can be alternatively achieved by a sequence of two s-expansions followed by an ℓ-expansion. The two s-expansions subdivide a "twice", i.e., a is replaced by two vertices z_1 and z_2, and arcs $a_1 = xz_1$, $a_2 = z_1z_2$, and $a_3 = z_2y$. Clearly, a_2' is a constriction, so ℓ-expansion applies to a_2'. This is equivalent to merging z_1 and z_2 into z, thereby making a_2 a loop.

It remains to show that for $G \Rightarrow_\triangleleft H$, $H' \in \mathbf{SPL}$ follows, too. Let y be the center vertex of an in-fan in G, i.e., assume that the sole in-arc of y is $a = xy$, and that $l_G(a) = \varepsilon$. Since the initial expression of the decoding is reduced, arcs with label ε either stem from sum decoding an arc with label $s^?$ or from star decoding. In the former case, however, there are at least two in-arcs of y, thus a is derived from star-decoding and x carries a loop. We can therefore assume $F \Rightarrow_* G \Rightarrow_\triangleleft H$, for an appropriate graph $F = (F', \mathcal{A}, l_F)$ with a vy-arc labeled s^*, as sketched below:



In total, this rewriting renames y to x an introduces a loop at x; also observe that all information about the labels of H are present in F already. The same effect can be achieved by s- and ℓ-expansion. First, s-expansion subdivides the vy-arc, replacing it with a new vertex x', and a vx'- and an $x'y$-arc. Since $\mathrm{d}^-_{F'}(y) = 1$, the $x'y$-arc is a constriction, ℓ-expansion applies next. We let x denote the merge vertex of x' and y to get exactly H'. This is shown below:



The argument is symmetric if an out-fan is considered. From $F' \in \mathbf{SPL}$, as implied by the inductive assumption, now follows $H' \in \mathbf{SPL}$. All decoding cases are hereby taken care of and the proof is complete. □

The fact that we get (labeled) spl-graphs from decoding an expression also applies to the structure of FAs constructed by $\mathfrak{B}$ the base ARS of the construction presented in Ch. 4.

Lemma 8.2.3. *Let r be an $\varnothing$-free expression. Then the graph structure of $A_r^{\mathfrak{B}}$ is in* **SPL**.

Proof. Let r be $\varnothing$-free. As in the proof of Cor. 4.3.3 we find $A_r^{\mathfrak{B}} = A_{r\bullet}^{\mathfrak{B}}$, so we can assume that r is in SSNF. Following Cor. 4.4.3, no ε-cycles appear in the construction of $A_r^{\mathfrak{B}}$, therefore, the only rules that are used in this construction are product, sum, and star expansion, together with fan elimination. The graph structure of the intermediate A_r^k is rewritten according to the decoding rules $\blacktriangleright.$, $\blacktriangleright_+$, $\blacktriangleright_*$, and $\blacktriangleright_\triangleleft$, so the claim follows from Prop. 8.2.2. □

The graph structure of constructed FAs has been investigated for other conversions, too. Caron & Ziadi [11] characterize the graph structure of Glushkov automata. Likewise, Giammarresi et al. [18] characterize the structure of FAs obtained by Thompson's construction.

8.3 Duality of Encoding and Decoding

Lemma 8.3.1.

1. *$G \blacktriangleleft^\star \mathsf{P}_1[r]$ implies $\mathsf{P}_1[r] \blacktriangleright^\star G$.*
2. *Let r be a reduced expression. Then $\mathsf{P}_1[r] \blacktriangleright^\star G$ implies $G \blacktriangleleft^\star \mathsf{P}_1[r']$ with $r =_{\mathrm{AC}} \mathrm{red}(r')$.*

Proof.

1. Let $G = (G', \mathcal{A}, l_G)$ and $H = (H', \mathcal{A}, l_H)$. To prove the claim, it is sufficient to show that any encoding step can be reverted by decodings, i.e., that $G \blacktriangleleft H$ implies $H \blacktriangleright^\star G$. The claim then follows from local confluence of the decoding relation (Lem. 8.2.1). We have seen in Ch. 6 that the dual relations of series, parallel and loop reductions are series, parallel, and loop expansions. Generalizing this to labeled graphs is straightforward, thus $G \blacktriangleleft. H$ implies $H \blacktriangleright. H$, $G \blacktriangleleft_+ H$ implies $H \blacktriangleright_+ H$, and $G \blacktriangleleft_* H$ implies $H \blacktriangleright_* H$.

2. We show that if $\mathsf{P}_1[r] \blacktriangleright^\star G$ and $G \blacktriangleright H$, for reduced r, then G and H are encoded as $G \blacktriangleleft^\star \mathsf{P}_1[r']$ and $H \blacktriangleleft^\star \mathsf{P}_1[r'']$ s.t. $\mathrm{red}(r') =_{\mathrm{AC}} \mathrm{red}(r'') =_{\mathrm{AC}} r$.

   For product and sum related rewritings the decoding and encoding relations are duals, as observed in the proof of the previous item. There, $G \blacktriangleright. H$ implies $H \blacktriangleleft. G$ and $G \blacktriangleright_+ H$ implies $H \blacktriangleleft. G$, and the claim follows immediately.

   For $G = (G', \mathcal{A}, l_G)$ and $H = (H', \mathcal{A}, l_H)$, consider the decoding step $G \blacktriangleright_* H$ and let $a = xy$ be the arc with $l_G(a) = s^*$ in G. The decoding replaces a with a vertex z and arcs $a_1 = xz$, $a_2 = zz$, and $a_3 = zy$, with labels $l_H = (a_1) = \varepsilon$, $l_H(a_2) = s$, and $l(a_3) = \varepsilon$. The loop a_2 allows for ℓ-reduction in H', so H allows for the corresponding star encoding. The resulting split vertices of z are both simple and thus allow for follow up product encodings. Therefore, the structure of G, is reinstated by a rewriting $H \blacktriangleleft_* G_1 \blacktriangleleft. G_2 \blacktriangleleft. G_3$, where $G_3 = (G', \mathcal{A}, l_{G_3})$, and l_{G_3} is identical to l_G, except that for the xy-arc we started with, we have $l_{G_3} = \varepsilon s^* \varepsilon$. The rewritings that lead from G to G_3 are sketched below:

   $$x \xrightarrow{s^*} y \quad \blacktriangleright_* \quad x \xrightarrow{\varepsilon} z \overset{s}{\circlearrowright} \xrightarrow{\varepsilon} y \quad \blacktriangleleft_* \quad x \xrightarrow{\varepsilon} z_1 \xrightarrow{s^*} z_2 \xrightarrow{\varepsilon} y \quad \blacktriangleleft^2_\cdot \quad x \xrightarrow{\varepsilon s^* \varepsilon} y$$

   Any expression that is the label of an arc in a labeled hammock occurs as a subexpression in the label of some arc in the $\mathfrak{E}$-normal form of that hammock. Since we assume $G' \in \mathbf{SPL}$, exhaustive encoding of G leads to $\mathsf{P}_1[r']$ with $\varepsilon s^* \varepsilon \in \mathrm{sub}(r')$. Reducing r' replaces $\varepsilon s^* \varepsilon$ with $\mathrm{red}(s^*)$; and since $s^* \in \mathrm{sub}(r)$ for reduced r by assumption, s^* is reduced already. Since we obtain $\mathrm{red}(r')$ as the encoding of G, the difference in the labeling of a in G and G_3 is irrelevant.

   For $G \blacktriangleright_\triangleleft H$ we assume that an in-fan is eliminated/decoded, the other case is symmetric. This case is similar to the last case of Prop. 8.2.2. The exact same argument as therein shows that the ε-arc in a fan results from

star expansion. Again, we may assume that if $a = xy$ with $l_G(a) = \varepsilon$ and $\mathrm{d}^-(y) = 1$, then x carries a loop a' with label s.

We apply star encoding to a' in G to get an x_1x_2-arc a''; the arc a with label ε now becomes an x_2y-arc. With x_2 being simple, we apply product encoding to get an x_1y-arc with label $s^*\varepsilon$. Let F_1 denote the resulting graph, i.e., let $G \blacktriangleleft_* \blacktriangleleft . F_1$. In H, star encoding a' yields an x_1x_2-arc labeled s^*, in the resulting graph F_2. Now F_1 and F_2 are identical up to the vertex name y, resp. x_2, and the label of the x_1y-arc, resp. the x_1x_2-arc. This label is s^*, resp. $s^*\varepsilon$, and by the same argument as in the previous case, we find the encodings $F_1 \blacktriangleleft^\star \mathsf{P}_1[r']$ and $F_2 \blacktriangleleft^\star \mathsf{P}_1[r'']$ with $\mathrm{red}(r') =_{\mathrm{AC}} \mathrm{red}(r'') =_{\mathrm{AC}} r$.

□

Of course, the first item in Lem. 8.3.1 is a statement about (labeled) spl-graphs, which are exactly the graphs that can be rewritten to the (labeled) axiom by (labeled) spl-reductions. In effect, the first item of the lemma thus states that any spl-graph can be reversibly encoded by an expression. The second item states that any reduced expression can be converted to a graph, from which the expression can be obtained again by reduction, modulo $=_{\mathrm{AC}}$, that is. We also know from Lem. 8.2.1 and Prop. 8.2.2 that any such graph is a uniquely determined (labeled) spl-graph.

Therefore, the encoding and decoding relations provide bijections between the class of spl-graphs, labeled or unlabeled, and that of reduced regular expression. As a corollary, we get the main result of this chapter, namely, that reduced expressions constitute a third characterization of spl-graph. The core of this statement is the preceding lemma, we give the following is a more "catchy" — and rather informal — rephrasing of these properties.

Theorem 8.3.2. *A graph G is an spl-graph iff G can be reversibly encoded by a reduced regular expression.*

As we have shown in Ch. 7, the class **SPL** admits a characterization by a finite set of forbidden (bare) minors. Moreover, we know that none of these minors can be omitted from the characterization. Since we know that that spl-graphs and FAs with spl-structure can be converted to linear sized expressions, our findings suggest that the forbidden minors of **SPL** constitute exactly the structural properties of general FAs that cannot be encoded efficiently by expressions. In other words, these are the structures that cause the exponential blowup generally observed with the conversion of FAs to expressions.

It is worth mentioning that the FAs which were used by Ehrenfeucht & Zeiger in the proof of this exponential lower bound have complete underlying graphs. In other words, there is a transition among each pair of states in either direction. For a fixed FA with n states of this type, it follows that every p-normal graph of order up to n contained in some way in this automaton. It should be interesting to investigate if the lower bound can be reinstated with less dense automata, that are built by members of **F** and **K** exclusively.

Bibliography

[1] Bang-Jensen, J., and Gutin, G. *Digraphs. Theory, Algorithms and Applications*, 2 ed. Springer Monographs in Mathematics. Springer, 2008.

[2] Becchi, M., and Crowley, P. A hybrid finite automaton for practical deep packet inspection. In *3rd Conference on emerging Networking EXperiments and Technologies* (2007).

[3] Bondy, J., and Murty, U. *Graph Theory*, 2 ed. No. 244 in Graduate Texts in Mathematics. Springer, 2008.

[4] Book, R. V., Even, S., Greibach, S., and Ott, G. Ambiguity in graphs and expressions. *IEEE Transactions on Computers 20*, 2 (1971), 149–153.

[5] Borie, R. B., Parker, R. G., and Tovey, C. A. Recursively constructed graphs. In *Handbook of Graph Theory*. CRC Press, 2004.

[6] Borie, R. B., Parker, R. G., and Tovey, C. A. Solving problems on recursively constructed graphs. *ACM Computing Surveys 41*, 1 (2008).

[7] Brandstädt, A., Le, V. B., and Spinrad, J. P. *Graph Classes: A Survey*. No. 3 in SIAM Monographs on Discrete Mathematics and Applications. 1999.

[8] Brüggemann-Klein, A. Regular expressions into finite automata. *Theoretical Computer Science 120* (1993), 197–312.

[9] Brüggemann-Klein, A., and Wood, D. One-unambiguous regular languages. *Information and Computation 142* (1998), 182–206.

[10] Brzozowski, J. A., and McCluskey Jr., E. J. Signal flow graph techniques for sequential circuit state diagrams. *IEEE Transactions on Electronic Computers 2* (1963), 67–76.

[11] Caron, P., and Ziadi, D. Characterization of Glushkov automata. *Theoretical Computer Science 233* (2000), 75–90.

[12] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*, 3 ed. The MIT Press, 2009.

[13] Diestel, R. *Graph Theory*, 3 ed. No. 173 in Graduate Texts in Mathematics. Springer, 2006.

[14] Ehrenfeucht, A., and Zeiger, P. Complexity measures for regular expressions. In *STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing* (New York, NY, USA, 1974), ACM Press, pp. 75–79.

[15] Ehrenfeucht, A., and Zeiger, P. Complexity measures for regular expressions. *Journal of Computer and System Sciences 12* (1976), 134–146.

[16] Ellul, K., Krawetz, B., Shallit, J., and Wang, M.-W. Regular expressions: new results and open problems. *Journal of Automata, Languages and Combinatorics 10*, 4 (2005), 407–437.

[17] Friedl, J. E. *Mastering Regular Expressions*, 3 ed. O'Reilly, 2006.

[18] Giammarresi, D., Ponty, J.-L., Wood, D., and Ziadi, D. A characterization of Thompson digraphs. *Discrete Applied Mathematics 134* (2004), 317–337.

[19] Ginzburg, A. A procedure for checking equality of regular expressions. *Journal of the ACM 14*, 2 (1967), 355–326.

[20] Granot, D., Granot, F., and Zhu, W. Naturally submodular digraphs and forbidden digraph configurations. *Discrete Applies Mathematics 100* (2000), 67–84.

[21] Gruber, H., and Gulan, S. Simplifying regular expressions. A quantitative perspective. In *LATA 2010* (2010), no. 6031 in LNCS, Springer.

[22] Gulan, S., and Fernau, H. An optimal construction of finite automata from regular expressions. In *28th International Conference on Foundations of Software Technology and Theoretical Computer Science* (2008), pp. 211–222.

[23] Hecht, M. S., and Ullman, J. D. Characterizations of reducible flow graphs. *Journal of the ACM 21*, 3 (1974), 367–375.

[24] Hemminger, R. L., and Klerlein, J. B. Line pseudographs. *Journal of Graph Theory 1* (1977), 365–377.

[25] Holzer, M., and Kutrib, M. Nondeterministic descriptional complexity of regular languages. *International Journal of Foundation of Computer Science 14*, 6 (2003), 1087–1102.

[26] Holzer, M., and Kutrib, M. Descriptional complexity — an introductory survey. In *Scientific Applications of Language Methods*, C. Martín-Vide, Ed. 2011.

[27] Hopcroft, J. E., and Ullman, J. D. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[28] Hovland, D. The inclusion problem for regular expressions. In *LATA 2010* (2010), no. 6031 in LNCS, Springer, pp. 309–320.

[29] Huet, G. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM 27*, 4 (1980), 797–821.

[30] Ilie, L., and Yu, S. Follow automata. *Information and Computation*, 186 (2003), 140–162.

[31] Kleene, S. C. *Representation of Events in Nerve Nets and Finite Automata*. Annals of Mathematics Studies. 1956, pp. 3–41.

[32] Kumar, S., Chandrasekaran, B., Turner, J., and Varghese, G. Curing regular expression matching algorithms from insomnia, amnesia and acalculia. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems* (2007), pp. 155–164.

[33] Kuratowski, C. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae 15* (1930), 271–283.

[34] Mehlhorn, K., and Näher, S. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[35] Mehlhorn, K., Näher, S., and Uhrig, C. The LEDA platform for combinatorial and geometric computing. In *ICALP* (1997), no. 1256 in LNCS, Springer, pp. 7–16.

[36] Meister, D., and Teller, J. A. Chordal digraphs. In *WG 2009* (2010), no. 5911 in LNCS, Springer, pp. 273–284.

[37] Meyer, A., and Stockmeyer, L. The equivalence problem for regular expressions with squaring requires exponential space. In *Proc. 13th Ann. IEEE Symp. on Switching and Automata Theory* (1972), pp. 125–129.

[38] Moreira, N., and Reis, R. Series-parallel automata and short regular expressions. *Fundamenta Informaticae 91*, 3-4 (2009), 611–629.

[39] Newman, M. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics 43*, 2 (1942).

[40] Ohlebusch, E. *Advanced Topics in Term Rewriting*. Springer, 2002.

[41] Ott, G., and Feinstein, N. H. Design of sequential machines from their regular expressions. *Journal of the ACM 8*, 4 (1961), 585–600.

[42] Oulsnam, G. Unravelling unstructured programs. *The Computer Journal 25*, 3 (1982), 379–387.

[43] Sakarovitch, J. The language, the expression, and the (small) automaton. In *CIAA 2005* (2006), no. 3845 in LNCS, Springer, pp. 15–30.

[44] Sakarovitch, J. *Elements of Automata Theory*. Cambridge University Press, 2009.

[45] Salomaa, A. Two complete axiom systems for the algebra of regular events. *Journal of the ACM 13*, 1 (1966), 158–169.

[46] Schoenmakers, B. A new algorithm for the recognition of series parallel graphs. Tech. Rep. CS-R9504, Centrum voor Wiskunde en Informatica, 1995.

[47] Shcherbina, O. A. Tree decomposition and discrete optimization problems: A survey. *Cybernetics and Systems Analysis 43*, 4 (2007), 549–562.

[48] Siek, J., Lee, L.-Q., and Lumsdaine, A. *The Boost Graph Library: User Guide and Reference Manual*. Pearson Education, 2002.

[49] Sippu, S., and Soisalin-Soininen, E. *Parsing Theory*. EATCS Monographs on Theoretical Computer Science. Springer, 1988.

[50] Sommer, R., and Paxson, V. Enhancing byte-level network intrusion detection signatures with context. In *Proc. 10th ACM Conference on Computer and Communications Security* (2003), pp. 262–271.

[51] Thompson, K. Regular expression search algorithm. *Communications of the ACM 11*, 6 (1968), 419–422.

[52] Valdes, J. *Parsing Flowcharts and Series-Parallel Graphs*. PhD thesis, Stanford University, 1978. STAN-CS-78-682.

[53] Valdes, J., Tarjan, R. E., and Lawler, E. L. The recognition of series parallel digraphs. In *Eleventh annual ACM symposium on Theory of computing* (1979), pp. 1–12.

[54] Valdes, J., Tarjan, R. E., and Lawler, E. L. The recognition of series parallel digraphs. *SIAM Journal on Computing 11*, 2 (1981), 298–313.

[55] Watson, B. W. A taxonomy of finite automata construction algorithms. Tech. Rep. Computing Science Note 93/43, Eindhoven University of Technology, May 1994.

[56] Yang, Y.-H. E., and Prasanna, V. K. Space-time tradeoff in regular expression matching with semi-deterministic finite automata. In *IEEE INFOCOM* (2011), pp. 1853 – 1861.

[57] Yu, S. Regular languages. In *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa, Eds. 1997, ch. 2.

Index